

Mathematics of Neural Networks

Volume IV: Advanced Optimization, Theory and Robustness

Loss landscape geometry, generalization bounds, and adversarial mathematics

Every derivation step by step

Every symbol defined

Explicit numerical examples

Geometric intuitions

Silvano Salvador

Contents

39.1	The Loss Surface as a Manifold	2
39.2	Saddle-Point Proliferation in High Dimensions	4
39.3	Flat Minima and Generalisation	6
39.5	Loss Landscape of Overparameterised Networks	8
39.4	Practical Hessian Estimation	10
39.6	The Edge of Stability and Catapult Dynamics	11
39.7	Loss Landscape of Transformer Architectures	12
40.1	Riemannian Geometry of the Parameter Space	13
40.2	Natural Gradient Update	14
40.3	K-FAC: Kronecker-Factored Approximate Curvature	16
40.4	Quasi-Newton Methods and L-BFGS	17
40.5	Shampoo and Distributed Second-Order Optimisation	19
41.1	The Generalisation Problem	21
41.2	PAC-Bayes Framework	24
41.3	Connection to Information Theory	26
41.4	Algorithmic Stability and Uniform Stability Bounds	28
41.5	Double Descent and the Interpolation Threshold	29
42.1	Mutual Information	31
42.2	The Information Bottleneck	32
42.3	Tishby's Neural Network Interpretation	34
42.4	Mutual Information Neural Estimation (MINE)	36
42.5	Variational Inference Meets Information Theory	37
43.1	The Adversarial Example Problem	39
43.2	Fast Gradient Sign Method (FGSM)	41
43.3	Projected Gradient Descent (PGD)	42
43.4	The Landscape of Adversarial Training	43
43.5	Black-Box Attacks and Score-Based Adversarial Examples	44

44.1	The Smoothed Classifier	47
44.2	Certification Theorem	48
44.3	Practical Certification via Monte Carlo	48
44.4	Certified Training and the SmoothAdv Objective	49
44.5	Interval Bound Propagation and Complete Verification	50
45.1	Differential Privacy	52
45.2	The Gaussian Mechanism	54
45.3	DP-SGD and the Moments Accountant	55
45.3b	Privacy Amplification by Subsampling	55
45.4	Federated Averaging (FedAvg)	56
45.5	Convergence Theory of Federated Averaging	56
46.1	Covariate Shift Formalisation	58
46.2	The Ben-David Bound	60
46.3	Domain-Adversarial Neural Network (DANN)	61
46.4	Self-Training and Pseudo-Labels for Domain Adaptation	62
46.5	FixMatch and Consistency Regularisation	63
46.6	Structured Covariate Shift and Joint Distribution Alignment	64
47.1	From ResNets to ODEs	66
47.2	The Adjoint Method	66
47.3	Continuous Normalising Flows	68
47.4	Physics-Informed Neural Networks (PINNs)	70
47.5	Diffusion Models as Stochastic Differential Equations	71
48.1	Optimal Control Formulation of Deep Learning	73
48.2	Pontryagin Maximum Principle	75
48.2b	Pontryagin Maximum Principle Applied to Deep Learning	75
48.3	Hamilton-Jacobi-Bellman Equation	76
48.4	Stochastic Optimal Control and Policy Gradient Methods	77
48.5	Hamiltonian Monte Carlo and Variational Inference	79

49.1	Continuous-Time Fourier Transform	81
49.2	Nyquist-Shannon Sampling Theorem	83
49.3	Aliasing in CNNs and Blur-Pooling	83
49.4	Spectral Analysis of Convolutional Architectures	84
50.1	Translation Equivariance of Continuous Convolution	87
50.2	Discrete Convolution and Circular Convolution	88
50.2b	The DFT Convolution Theorem and Efficient Computation	88
50.3	Steerable Filters and $SE(2)$ Equivariance	89
50.4	Spectral Graph Convolution and Graph Neural Networks	90
50.5	Equivariant Neural Networks and Group Theory	91
51.1	The Adversarial Training Objective	94
51.2	The Robustness-Accuracy Trade-off	94
51.3	Distributionally Robust Optimisation (DRO)	94
51.4	Connections between Robustness Frameworks	96
52.1	Core Results of Volume IV	97
52.2	Mathematical Unification: Geometry, Information, and Robustness	98
52.3	Key Theorems and Results in Volume IV	99
52.4	Bridges to Volume V	100
	Dimostrazioni Differite	102
	Ricettario di Ottimizzazione Avanzata e Robustezza	104
	Complessità e Tradeoff Computazionali	105
	Glossario dei Simboli — Volume IV	106
	Bibliografia	108

Loss Landscape Geometry: Critical Points and Saddle Escape

We analyse the geometry of the loss surface $\mathcal{L} : \mathbb{R}^d \rightarrow \mathbb{R}$, classify critical points via the Hessian spectrum, and show why high-dimensional saddles — not local minima — are the dominant obstacle for gradient descent.

39.1 The Loss Surface as a Manifold

The loss function $\mathcal{L} : \mathbb{R}^d \rightarrow \mathbb{R}$ of a deep neural network defines a hypersurface in $(d + 1)$ -dimensional space. For modern networks with $d \sim 10^8$ parameters, this surface is an extraordinarily high-dimensional object whose geometry determines the entire optimisation trajectory.

Loss Landscape Vocabulary

- **Critical point:** $\nabla_{\square} \mathcal{L} = \mathbf{0}$. Includes minima, maxima, and saddle points.
- **Local minimum:** $\nabla^2 \mathcal{L} \succ 0$ (positive definite Hessian).
- **Saddle point:** Hessian has both positive and negative eigenvalues.
- **Degenerate saddle:** One or more zero eigenvalues of the Hessian.
- **Loss basin:** A connected region where $\mathcal{L}(\square) \leq \mathcal{L}^* + \epsilon$ for some $\epsilon > 0$.

Manifold structure. The sublevel set $\mathcal{S}_c = \{\square : \mathcal{L}(\square) \leq c\}$ is a manifold (with boundary) when c is a regular value of $\mathcal{L}$ (i.e., no critical point has $\mathcal{L} = c$). By Morse theory, the topology of $\mathcal{S}_c$ changes precisely when c passes through a critical value. Each saddle point with k negative Hessian eigenvalues adds a k -handle to the topology of $\mathcal{S}_c$.

Intrinsic dimension and effective dimensionality. Although $\square \in \mathbb{R}^d$, the loss surface often has an effective intrinsic dimension much smaller than d . Experiments show that a random subspace of dimension ~ 200 – 2000 can contain near-optimal solutions for ImageNet-scale problems. This *intrinsic dimensionality* is defined as the smallest k such that a k -dimensional random affine subspace contains a solution with performance within ϵ of the full-dimensional optimum.

EXAMPLE — Visualising a 2D Slice

Given a trained parameter vector $\square^*$ and two random unit vectors $\mathbf{u}, \mathbf{v} \in \mathbb{R}^d$, define:

$$f(\alpha, \beta) = \mathcal{L}(\square^* + \alpha \mathbf{u} + \beta \mathbf{v})$$

Filter normalisation ensures scale-invariance: each direction $\mathbf{u}$ is scaled so that $\|u_l\|_F = \|\theta_l^*\|_F$ for each layer l . Without this normalisation, different architectures produce incomparable curvatures.

Overparameterisation and flat directions. When the number of parameters d greatly exceeds the number of training points n , the loss surface generically has a $d - n$ dimensional *flat manifold* of global minima. Every point on this manifold achieves zero training loss, but they differ dramatically in their generalisation properties. The geometry of this manifold — specifically its curvature as measured by the Hessian restricted to the manifold — determines which solutions are found by gradient descent.

Mode connectivity. A remarkable empirical discovery (Garipov et al. 2018) is that distinct local minima found by independent training runs can be connected by low-loss paths in

parameter space. These paths are not straight lines but piecewise-linear or quadratic Bézier curves. This *mode connectivity* suggests that the loss landscape, despite appearing to have many isolated minima, actually has a highly connected basin structure at the global scale.

The distinction between *sharp* and *flat* minima (§39.3) is not scale-invariant under reparametrisation. A minimum that appears sharp in one coordinate system may appear flat in another. Sharpness-aware measures like the trace of the Hessian must be interpreted with care, and filter-normalised metrics are generally preferred in practice.

Topology changes at critical values. By the fundamental theorem of Morse theory, the homotopy type of the sublevel set $\mathcal{S}_c = \{\square : \mathcal{L}(\square) \leq c\}$ changes precisely at critical values. Passing a saddle of index k adds a k -cell (a k -dimensional handle) to the topology. Concretely, for a 2-layer network:

- Passing a local minimum ($k = 0$): adds a new connected component (a new basin).
- Passing a saddle ($k = 1$): merges two existing components by a “tube”.
- Passing a local maximum ($k = d$): closes off an existing component.

The Morse inequalities constrain the number of critical points: $\#\{\text{index-}k \text{ critical points}\} \geq b_k$ where b_k are the Betti numbers of $\mathcal{S}_c$. For overparameterised networks, the sublevel sets at the global minimum are likely connected (no isolated basins), explaining why gradient descent can reach any minimum.

Energy barriers. The height of the energy barrier between two minima $\square_1, \square_2$ is:

$$\Delta E(\square_1, \square_2) = \min_{\gamma: \square_1 \rightarrow \square_2} \max_t \mathcal{L}(\gamma(t)) - \frac{\mathcal{L}(\square_1) + \mathcal{L}(\square_2)}{2}$$

where the minimum is over all continuous paths γ . Garipov et al. (2018) showed empirically that $\Delta E \approx 0$ for pairs of independently trained ResNets — there exist nearly flat connecting curves, making the loss surface much more connected than it appears.¹

Loss landscape visualisation via random subspaces. Goodfellow et al. (2015) observed that the one-dimensional slice $\mathcal{L}(\square^* + \alpha(\square^{(1)} - \square^{(2)}))$ between two independently trained minima is often nearly flat, with only small bumps. This contrasts with a naive expectation of high barriers between isolated minima. The phenomenon holds even for deep networks with millions of parameters.

EXAMPLE — Mode Connectivity on CIFAR-10 ResNet-20

Two ResNet-20 models trained from different random initialisations:

- Model A: test loss 0.412, accuracy 92.1%.
- Model B: test loss 0.419, accuracy 91.9%.

Linear interpolation $\square(\alpha) = (1 - \alpha)\square_A + \alpha\square_B, \alpha \in [0, 1]$: maximum test loss along the line is 1.82 (the midpoint is in a high-loss region).

Quadratic Bézier curve $\square(\alpha) = (1 - \alpha)^2\square_A + 2\alpha(1 - \alpha)\square_{\text{mid}} + \alpha^2\square_B$ with $\square_{\text{mid}}$ optimised: maximum test loss along the curve is 0.427 — nearly flat.

This confirms that the two minima are connected by a low-loss path in parameter space, though not a straight line. The “loss canyon” is a curved submanifold, not an

¹This “mode connectivity” has been verified across architectures (ResNets, Transformers) and datasets (CIFAR-10, ImageNet).

isolated basin.

For a neural network with d parameters $\square \in \mathbb{R}^d$, the loss function $\mathcal{L}(\square)$ defines a hypersurface in $\mathbb{R}^{d+1}$. A *critical point* is any $\square^*$ where $\nabla \mathcal{L}(\square^*) = \mathbf{0}$.

Classification of Critical Points

Let $\mathbf{H}(\square^*) = \nabla^2 \mathcal{L}(\square^*)$ be the Hessian at a critical point.

- **Local minimum:** all eigenvalues $\lambda_i > 0$ (positive definite).
- **Local maximum:** all $\lambda_i < 0$ (negative definite).
- **Saddle point:** mixed signs. The *index* of the saddle is the number of negative eigenvalues.
- **Degenerate (flat) critical point:** some $\lambda_i = 0$.

Second-order Taylor expansion at a critical point. Since $\nabla \mathcal{L}(\square^*) = \mathbf{0}$, the local behaviour is determined entirely by the Hessian:

$$\mathcal{L}(\square^* + \square) = \mathcal{L}(\square^*) + \frac{1}{2} \square^\top \mathbf{H}(\square^*) \square + O(\|\square\|^3)$$

Expanding in the eigenbasis $\mathbf{H}\mathbf{v}_i = \lambda_i \mathbf{v}_i$, with $\square = \sum_i a_i \mathbf{v}_i$:

$$\mathcal{L}(\square^* + \square) \approx \mathcal{L}(\square^*) + \frac{1}{2} \sum_i \lambda_i a_i^2$$

Each direction $\mathbf{v}_i$ is independent. The loss increases along directions with $\lambda_i > 0$ (stable) and decreases along directions with $\lambda_i < 0$ (unstable escape directions).

Morse lemma. If $\square^*$ is a non-degenerate critical point (all $\lambda_i \neq 0$), there exists a smooth coordinate change $\square$ near $\square^*$ such that:

$$\mathcal{L}(\square) = \mathcal{L}(\square^*) - \phi_1^2 - \dots - \phi_k^2 + \phi_{k+1}^2 + \dots + \phi_d^2$$

where k is the Morse index (number of negative eigenvalues). This is the canonical local form: the loss surface is topologically equivalent to a standard saddle of index k near any non-degenerate critical point.

EXAMPLE — Morse Index and Local Geometry in 3D

Consider $\mathcal{L}(\theta_1, \theta_2, \theta_3) = \theta_1^2 + \theta_2^2 - \theta_3^2$. At the origin:

$$\mathbf{H} = \text{diag}(2, 2, -2), \quad \lambda_1 = \lambda_2 = 2 > 0, \quad \lambda_3 = -2 < 0$$

Morse index $k = 1$ (one negative eigenvalue). The loss has a “valley” in the (θ_1, θ_2) -plane and a “ridge” in the θ_3 -direction. Gradient descent escapes along $-\mathbf{e}_3$ if started with $\theta_3 \neq 0$, and converges in (θ_1, θ_2) .

For a network with $d = 10^7$: a saddle of index 5×10^6 has 5 million escape directions. Any algorithm that adds even a small amount of noise along any of these directions will escape efficiently.

39.2 Saddle-Point Proliferation in High Dimensions

Consider a random function where each eigenvalue of the Hessian is independently $+1$ or -1 with equal probability. The probability that a critical point is a local minimum (all d eigenvalues positive) is 2^{-d} . For $d = 10^7$ parameters, this probability is astronomically small.

The saddle dominance principle. In high dimensions, almost all critical points are saddles with approximately $d/2$ negative eigenvalues. The loss value at a saddle is typically *higher* (worse) than at a true minimum, so the loss at a saddle correlates with the saddle index.

EXAMPLE — Saddle on a 2D loss surface

Consider $\mathcal{L}(\theta_1, \theta_2) = \theta_1^2 - \theta_2^2$. At the origin:

$$\nabla \mathcal{L} = (2\theta_1, -2\theta_2)^\top \Big|_0 = \mathbf{0}$$

$$\mathbf{H} = \begin{pmatrix} 2 & 0 \\ 0 & -2 \end{pmatrix}$$

Eigenvalues $+2$ (direction θ_1) and -2 (direction θ_2). This is a saddle of index 1. Along the θ_1 -axis: $\mathcal{L}$ is minimised (curvature up). Along the θ_2 -axis: $\mathcal{L}$ is maximised (curvature down). SGD noise escapes along the negative-curvature direction.

Escape from saddles: gradient noise mechanism. Gradient descent (without noise) can be trapped at a saddle indefinitely if started exactly at the saddle. The gradient is zero, so no update occurs. In practice, two mechanisms cause escape:

1. **Floating-point noise:** accumulation of rounding errors in 32-bit arithmetic adds $O(10^{-7})$ perturbations per step. For a saddle with $\lambda_{\min} = -\lambda < 0$, the perturbation along the escape direction grows as $e^{\lambda t}$, so escape occurs in $O(\log(1/\epsilon)/\lambda)$ steps where $\epsilon \sim 10^{-7}$.
2. **Mini-batch gradient noise:** the stochastic gradient estimator has variance $O(\sigma^2/B)$ where B is the batch size. The component along the escape direction has standard deviation $\sim \sigma/\sqrt{B}$, providing a consistent noise injection.

Quantitative escape time. Jin et al. (2017) prove that perturbed gradient descent (PGD) — gradient descent with added isotropic Gaussian perturbation $\mathcal{N}(0, r^2 \mathbf{I})$ every f_{thres} steps — escapes all strict saddles and converges to an approximate second-order stationary point in $O(1/\epsilon^2 + 1/\epsilon^4 \epsilon_H)$ gradient evaluations where ϵ_H is the negative curvature threshold.

Negative curvature exploitation. An alternative to noise injection is to explicitly follow negative-curvature directions. Given the escape direction $\mathbf{v}_{\min}$ (eigenvector of the Hessian with eigenvalue $\lambda_{\min} < 0$), the update:

$$\square \leftarrow \square - \eta_{\text{escape}} \text{sign}(\mathbf{v}_{\min}^\top \nabla^2 \mathcal{L} \mathbf{v}_{\min}) \mathbf{v}_{\min}$$

takes a step along the descent direction for the saddle. In practice, $\mathbf{v}_{\min}$ is approximated by a few steps of inverse power iteration (Lanczos) or by using the negative curvature information from the Hessian-vector product.

Perturbed Gradient Descent (PGD)

Algorithm (Jin et al., 2017):

1. Run gradient descent for f_{thres} steps.
2. If $\|\nabla \mathcal{L}\| \leq \epsilon_g$ (near-stationary), add perturbation $\square \sim \mathcal{N}(0, r^2 \mathbf{I})$ with $r = \delta$ (escape radius).
3. Continue for t_{escape} steps; if loss decreases by $\geq f_{\text{thres}} \cdot \chi$, the perturbation succeeded. Otherwise, declare convergence.

Guarantee: Converges to (ϵ_g, ϵ_H) -SOSP in $O\left(\frac{\ell^2 \Delta_{\mathcal{L}}}{\epsilon_g^4} \log \frac{d}{\delta}\right)$ iterations, where ℓ is the Hessian Lipschitz constant and $\Delta_{\mathcal{L}}$ is the initial loss gap.

Saddle-free Newton. Dauphin et al. (2014) propose a second-order method that removes saddle directions by replacing $\mathbf{H}$ with $|\mathbf{H}| = \sum_i |\lambda_i| \mathbf{v}_i \mathbf{v}_i^\top$:

$$\square_{t+1} = \square_t - \eta |\mathbf{H}(\square_t)|^{-1} \nabla \mathcal{L}(\square_t)$$

This “trust region” update moves in the direction of steepest descent *in the eigenspace of positive curvature* and in the anti-gradient direction in negative-curvature directions, explicitly escaping saddles rather than just perturbing around them. The cost is computing the eigen-decomposition of $\mathbf{H}$, which is $O(d^3)$ in general (reduced to $O(d)$ with stochastic Lanczos approximation).

Strict saddle property. A function satisfies the *strict saddle* property if every saddle point has at least one strictly negative Hessian eigenvalue ($\lambda_{\min} < 0$). For strict saddle functions, perturbed gradient descent escapes saddles in polynomial time with high probability (Jin et al., 2017). Neural network loss surfaces are believed to satisfy the strict saddle property generically (except at degenerate points corresponding to neurons with zero activation across all training examples).

First-Order Stationary Points and Second-Order Optimality

- **First-order stationary (FOSP):** $\|\nabla \mathcal{L}\| \leq \epsilon_g$.
- **Second-order stationary (SOSP):** FOSP and $\lambda_{\min}(\mathbf{H}) \geq -\epsilon_H$.
- **Local minimum:** SOSP with $\epsilon_H = 0$.

Gradient descent converges to FOSPs. Perturbed gradient descent (Jin et al., 2017) converges to SOSPs in $O(1/\epsilon_g^2 + 1/\epsilon_g^4 \epsilon_H)$ iterations — escaping from saddles with negative curvature $\leq -\epsilon_H$.

39.3 Flat Minima and Generalisation

A flat minimum is a region where $\mathcal{L}(\square) \approx \mathcal{L}(\square^*)$ for $\square$ in a large neighbourhood. The sharpness of a minimum is quantified by the trace or maximum eigenvalue of the Hessian:

$$\text{Sharpness} = \lambda_{\max}(\mathbf{H}), \quad \text{Flatness} = -\lambda_{\max}(\mathbf{H})$$

Why flatness correlates with generalisation. Hochreiter & Schmidhuber (1997) first conjectured the flatness-generalisation link using minimum-description-length arguments: a flat minimum can be described with fewer bits (the weights need less precision) than a sharp one. Keskar et al. (2017) showed empirically that large-batch SGD converges to sharp minima with poor generalisation, while small-batch SGD finds flat minima. The intuitive reason: large-batch gradients are more accurate and follow steep descent directions, landing in narrow basins; small-batch noise acts as implicit regularisation.

Sharpness Measures

- **$\lambda_{\max}$ -sharpness:** $\lambda_{\max}(\nabla^2 \mathcal{L}(\square))$. Sensitive to outlier curvature directions.
- **Trace sharpness:** $\text{tr}(\nabla^2 \mathcal{L}(\square)) = \sum_i \lambda_i$. Averages over all directions.
- **ϵ -sharpness** (Keskar et al.): $\max_{\|\square\|_\infty \leq \epsilon} \mathcal{L}(\square + \square) - \mathcal{L}(\square)$.
- **PAC-Bayes sharpness:** $\text{KL}(Q_\sigma \| P)$ where $Q_\sigma = \mathcal{N}(\square, \sigma^2 \mathbf{I})$.

The Hessian spectral structure. For overparameterised networks, the Hessian $\mathbf{H} = \nabla^2 \mathcal{L}$ has a characteristic bulk-outlier structure:

- A *bulk* of near-zero eigenvalues, corresponding to directions in parameter space that barely affect the loss (the flat manifold of solutions).
- A few *outlier* eigenvalues of large magnitude, corresponding to sensitive directions. These are associated with task-relevant features.

The ratio $\lambda_{\max}/\text{tr}(\mathbf{H})$ measures the relative importance of the most sensitive direction. For well-trained modern networks, this ratio is often surprisingly small, indicating distributed sensitivity.

Implicit bias of gradient descent. It is now rigorously established that gradient flow on the squared loss for linear networks converges to the *minimum-norm* solution — the flattest minimum in the sense of minimal ℓ_2 norm. For deep linear networks, gradient flow converges to the solution minimising the nuclear norm of the end-to-end weight matrix. These results provide a theoretical foundation for the flatness-generalisation hypothesis in the linear case.

EXAMPLE — Sharpness Measurement in Practice

For a ResNet-50 trained on CIFAR-10, the Hessian spectral density (computed via stochastic Lanczos quadrature) typically shows:

- 99% of eigenvalues in $[-0.1, 0.1]$ (the bulk).
- Top eigenvalue $\lambda_1 \approx 20\text{--}100$ depending on batch size.
- Large-batch ($B = 4096$): $\lambda_1 \approx 80$, test accuracy 92%.
- Small-batch ($B = 128$): $\lambda_1 \approx 15$, test accuracy 94%.

The 6-percentage-point accuracy gap corresponds to a $5\times$ difference in sharpness.

Flatness is not scale-invariant: multiplying a layer's weights by α and dividing the next layer's weights by α preserves the function but changes the Hessian spectrum. Filter-normalised sharpness, which normalises each weight direction by the corresponding filter norm, provides a scale-invariant alternative.

Sharpness-Aware Minimisation (SAM). Foret et al. (2021) propose directly minimising a flatness measure as part of training. The SAM objective is:

$$\min_{\square} \max_{\|\square\|_2 \leq \rho} \mathcal{L}(\square + \square)$$

This seeks parameters where the loss is low *and* stays low in a neighbourhood of radius ρ .

SAM gradient computation. The gradient of the outer minimum requires the inner maximum, approximated by one gradient ascent step:

$$\hat{\square} = \rho \cdot \frac{\nabla_{\square} \mathcal{L}(\square)}{\|\nabla_{\square} \mathcal{L}(\square)\|_2}$$

The SAM update is then:

$$\square_{t+1} = \square_t - \eta \cdot \nabla_{\square} \mathcal{L}(\square_t + \hat{\square})$$

This requires two gradient evaluations per step (one for $\hat{\square}$, one for the update), doubling the compute cost but consistently finding flatter minima.

EXAMPLE — SAM on CIFAR-10 ResNet-56

Training for 200 epochs with SGD and SAM (same schedule):

Method	Test Accuracy	$\lambda_{\max}(\mathbf{H})$
SGD (batch 128)	93.1%	18.3
AdamW	92.8%	21.7
SAM ($\rho = 0.05$)	94.6%	6.1
SAM ($\rho = 0.1$)	94.9%	3.8

SAM reduces $\lambda_{\max}$ by $5\times$ while improving test accuracy by 1.8 percentage points. The sharpness reduction directly translates to better generalisation, confirming the flatness-generalisation connection empirically.

A flat minimum is a region where $\mathcal{L}(\square) \approx \mathcal{L}(\square^*)$ for $\square$ in a large neighbourhood. The sharpness of a minimum is quantified by the trace or maximum eigenvalue of the Hessian:

$$\text{Sharpness} = \lambda_{\max}(\mathbf{H}), \quad \text{Flatness} = -\lambda_{\max}(\mathbf{H})$$

Flatness and generalisation. Sharp minima generalise poorly: a small shift in the data distribution moves the true optimum, and the sharp minimum has high loss there. Flat minima are robust to such shifts. Empirically, SGD with large learning rate finds flatter minima than full-batch GD, partly explaining the generalisation advantage of SGD.

39.5 Loss Landscape of Overparameterised Networks

The overparameterised regime ($d \gg n$) exhibits qualitatively different landscape behaviour from the classical regime. Understanding this regime is essential for modern deep learning.

The manifold of global minima. When $d > n$ (more parameters than data points), the training loss $\mathcal{L}(\theta) = 0$ is typically achievable by many distinct parameter vectors. The set of global minima:

$$\mathcal{M}^* = \{\theta : \mathcal{L}(\theta) = 0\}$$

is generically a smooth manifold of dimension $d - n$ (assuming the Jacobian $\mathbf{J}_\theta = \partial f_\theta(\mathbf{X}) / \partial \theta$ has full row rank n at each solution). The Hessian restricted to $\mathcal{M}^*$ has $d - n$ zero eigenvalues (corresponding to directions along the manifold) and n positive eigenvalues (directions away from it).

Gradient flow dynamics on the solution manifold. Starting from a random initialisation and running gradient flow $\dot{\theta} = -\nabla \mathcal{L}(\theta)$, the trajectory converges to the point on $\mathcal{M}^*$ that minimises the distance from the initialisation (in the Euclidean norm for linear networks, and in more complex norms for nonlinear ones). This is the *implicit bias* of gradient descent:

Implicit Bias of Gradient Descent (Linear Networks)

For a depth- L linear network $f_\theta(\mathbf{x}) = \mathbf{W}_L \cdots \mathbf{W}_1 \mathbf{x}$ trained with squared loss and gradient flow starting from $\theta(0) = \epsilon \mathbf{I}$ (small isotropic initialisation), the limiting solution is:

$$\mathbf{W}_\infty = \arg \min_{\mathbf{W} : \mathbf{W}\mathbf{X} = \mathbf{Y}} \|\mathbf{W}\|_*$$

where $\|\cdot\|_*$ is the nuclear norm (sum of singular values). This is the minimum-nuclear-norm interpolating solution, which is also the maximum-rank-margin solution.

Proof sketch. In the gradient flow limit, the singular values σ_i of the end-to-end matrix $\mathbf{W} = \mathbf{W}_L \cdots \mathbf{W}_1$ evolve as $\dot{\sigma}_i = -\partial \mathcal{L} / \partial \sigma_i$. With balanced initialisation ($\|\mathbf{W}_l\|_F = \text{const}$), the dynamics conserve $\sigma_i^{2/L}$, and the nuclear norm is decreased monotonically. The limiting solution is the minimum nuclear norm interpolator.

Implications for generalisation. The minimum nuclear norm solution is a low-rank approximation of the data:

- *Low rank:* many singular values of $\mathbf{W}_\infty$ are near zero, corresponding to a compressed representation.
- *Smooth interpolation:* the nuclear norm minimum satisfies a maximum-margin condition in function space, giving smooth predictions between training points.
- *Benign overfitting:* despite interpolating training data, the minimum-norm solution can have small test error when the data covariance has many small eigenvalues (“spiked covariance” structure), as shown by Bartlett et al. (2020).

The neural tangent kernel (NTK) regime. For sufficiently wide networks, the loss landscape becomes nearly convex in a neighbourhood of the initialisation. Define the NTK matrix:

$$\mathbf{K}_\theta(\mathbf{x}_i, \mathbf{x}_j) = \nabla_\theta f_\theta(\mathbf{x}_i)^\top \nabla_\theta f_\theta(\mathbf{x}_j)$$

In the infinite-width limit, $\mathbf{K}_\theta \rightarrow \mathbf{K}_{\text{NTK}}$ (a fixed kernel), and gradient descent on the squared loss converges to:

$$f_{\theta_\infty}(\mathbf{x}) = \mathbf{K}_{\text{NTK}}(\mathbf{x}, \mathbf{X}) \mathbf{K}_{\text{NTK}}(\mathbf{X}, \mathbf{X})^{-1} \mathbf{Y}$$

— kernel regression with the NTK kernel. This is a global minimum, and gradient descent finds it in $O(\log(1/\varepsilon)/\lambda_{\min}(\mathbf{K}_{\text{NTK}}))$ steps. The NTK approximation breaks down for finite-width networks and feature learning tasks.

EXAMPLE — NTK Eigenspectrum and Convergence Speed

For a 2-hidden-layer ReLU network, width $m = 1000$, trained on CIFAR-10 with $n = 50000$ examples:

- NTK matrix $\mathbf{K} \in \mathbb{R}^{50000 \times 50000}$ (not computed explicitly).
- Eigenspectrum: bulk of eigenvalues near 0 (as $m \rightarrow \infty$, the bulk collapses); top eigenvalues $\lambda_1 \approx 100$, $\lambda_{50000} \approx 10^{-3}$.
- Gradient flow convergence rate: $e^{-\lambda_{\min}(\mathbf{K})t}$. For $\lambda_{\min} \approx 10^{-3}$: convergence in $\sim 10^3 / \lambda_{\min} = 10^6$ steps.
- Practical width $m = 100$ (finite): top eigenvalues fluctuate by $\pm 30\%$, features *do* learn, NTK approximation fails.

The NTK regime applies to very wide networks but breaks down for practical architectures. The “edge of the NTK regime” is where the most interesting learning (and best generalisation) occurs.

Natural Gradient and the Fisher–Rao Metric

Ordinary gradient descent ignores the geometry of the parameter space. Natural gradient descent uses the Fisher information matrix to follow the steepest descent in distribution space, not Euclidean parameter space.

39.4 Practical Hessian Estimation

Computing the full Hessian $\mathbf{H} \in \mathbb{R}^{d \times d}$ is infeasible for large networks. Several practical methods estimate spectral properties without forming $\mathbf{H}$ explicitly.

Hessian-vector products. The key primitive is computing $\mathbf{H}\mathbf{v}$ for a given vector $\mathbf{v}$ in $O(P)$ time (same as a gradient computation):

$$\mathbf{H}\mathbf{v} = \nabla_{\theta}(\nabla_{\theta}\mathcal{L})^{\top}\mathbf{v} = \frac{d}{d\epsilon}\nabla_{\theta}\mathcal{L}(\theta + \epsilon\mathbf{v})\big|_{\epsilon=0}$$

This is computed via two backward passes: one to get $\nabla_{\theta}\mathcal{L}$, then differentiate through it with respect to $\mathbf{v}$ (forward-over-backward AD).

Power iteration for $\lambda_{\max}$. Repeated multiplication by $\mathbf{H}$ converges to the top eigenvector:

1. Initialise $\mathbf{v}_0 \sim \mathcal{N}(0, \mathbf{I})$.
2. Iterate: $\mathbf{v}_{k+1} = \mathbf{H}\mathbf{v}_k / \|\mathbf{H}\mathbf{v}_k\|$.
3. After convergence: $\lambda_{\max} \approx \mathbf{v}_k^{\top}\mathbf{H}\mathbf{v}_k$.

Convergence rate: $|\lambda_2/\lambda_1|^k$ per iteration. For a well-separated top eigenvalue, 20–50 iterations suffice.

Stochastic Lanczos quadrature. Computes the spectral density $\rho(\lambda) = \frac{1}{d} \sum_i \delta(\lambda - \lambda_i)$ approximately:

1. Draw $\mathbf{v} \sim \mathcal{N}(0, \mathbf{I}/d)$.
2. Compute the Lanczos decomposition: run k steps of the Lanczos algorithm with starting vector $\mathbf{v}$, producing a tridiagonal matrix $\mathbf{T}_k \in \mathbb{R}^{k \times k}$.
3. Estimate $\rho(\lambda) \approx \sum_{j=1}^k |\tau_{j1}|^2 \delta(\lambda - \lambda_j(\mathbf{T}_k))$, where τ_{j1} are the first components of the eigenvectors of $\mathbf{T}_k$.

This gives a smooth approximation of the spectral density with $O(k)$ Hessian-vector products. For $k = 100$ and a network with $d = 10^7$: 100 backward passes, producing a 100×100 tridiagonal system whose eigenvalues approximate the full Hessian’s spectral density.

EXAMPLE — Hessian Spectral Density of a ResNet-20 on CIFAR-10

Computed via stochastic Lanczos ($k = 90$ steps, 5 probe vectors, averaged).
 Bulk eigenvalue mass: 98.7% in $[-0.5, 0.5]$ (flat manifold of solutions).
 Outlier eigenvalues: 5 eigenvalues in $[15, 85]$, corresponding to the most sensitive weight directions (typically the last layer and batch norm parameters).
 Trace (sum of eigenvalues): ≈ 150 (computed as $\mathbb{E}_{\mathbf{v}}[\mathbf{v}^{\top}\mathbf{H}\mathbf{v}]$ with 10 random vectors).
 Top eigenvalue: $\lambda_{\max} \approx 82$ (via 30 power iterations). Condition number $\approx 82/0.001 = 82000$ (ratio of max to min non-zero eigenvalue).

39.6 The Edge of Stability and Catapult Dynamics

Recent empirical and theoretical work has uncovered surprising phenomena in the loss landscape during large-learning-rate training: the network operates at the “edge of stability” where the Hessian spectral norm oscillates near the stability threshold.

The progressive sharpening phenomenon. Cohen et al. (2021) observe that when training neural networks with gradient descent (full-batch, no noise), the maximum Hessian eigenvalue $\lambda_1(t)$ grows progressively during training until it reaches $2/\eta$ (the stability threshold of gradient descent), then oscillates around this value. This is called the *edge of stability* (EoS).

Stability threshold derivation. For gradient descent $\theta_{t+1} = \theta_t - \eta \nabla \mathcal{L}(\theta_t)$ applied to a quadratic $\mathcal{L}(\theta) = \frac{1}{2} \theta^\top \mathbf{H} \theta$, the update in the eigenbasis of $\mathbf{H}$ decouples:

$$a_i(t+1) = (1 - \eta \lambda_i) a_i(t)$$

This is stable iff $|1 - \eta \lambda_i| < 1$, i.e. $\lambda_i < 2/\eta$. The threshold $2/\eta$ is the stability boundary.

Catapult dynamics. When $\lambda_1 > 2/\eta$, gradient descent is unstable in the top eigendirection. Instead of diverging, the network exhibits catapult dynamics (Lewkowycz et al., 2020):

1. *Catapult phase:* $\lambda_1 > 2/\eta$; the loss oscillates or temporarily increases.
2. *Resolved phase:* λ_1 drops below $2/\eta$ (the sharpness is reduced); training stabilises.
3. *Convergence:* $\lambda_1 \leq 2/\eta$ is maintained; the network converges to a flat minimum.

The catapult mechanism is a *sharpness reduction* effect: large gradient steps in the top eigendirection overshoot and land in regions of lower sharpness. This is an implicit form of SAM (§39.3): large learning rates automatically find flatter minima.

Edge of Stability Dynamics

For a two-layer linear network $\mathcal{L}(\mathbf{W}) = \frac{1}{2} \|\mathbf{W}\mathbf{X} - \mathbf{Y}\|_F^2$, gradient descent with step size η satisfies:

- $\lambda_1(\mathbf{H}(t)) \rightarrow 2/\eta$ as $t \rightarrow \infty$ (edge of stability convergence).
- The trajectory along the top eigenvector oscillates with period $\approx 2\pi/\sqrt{\lambda_1 - 2/\eta}$.
- The loss decreases monotonically in other eigendirections.
- The final minimum has $\lambda_1 = 2/\eta$ exactly (a “sharp” minimum relative to the step size, but the sharpest achievable minimum at that step size).

Implications for learning rate scheduling. The edge of stability perspective explains why:

- Large learning rates find flatter minima (catapult mechanism reduces sharpness).
- Learning rate warmup (gradually increasing η) avoids early catapult instability.
- Learning rate decay (cosine schedule) allows the network to refine the minimum found by the large-LR catapult phase.
- The “linear warmup to $\eta_{\max}$, cosine decay to 0” schedule used in Transformers is a principled implementation of: (1) stabilise training, (2) catapult to a flat minimum, (3) refine.

EXAMPLE — Edge of Stability on ResNet-56

Training ResNet-56 on CIFAR-10 with gradient descent (no momentum), learning rate $\eta = 0.1$.

Stability threshold: $2/\eta = 20$.

Observed sharpness trajectory:

Epoch	$\lambda_1(\mathbf{H})$	Training loss
0	1.2	2.31
5	8.4	1.42
10	18.3	0.85
15	22.1 ($> 2/\eta$!)	0.93 (increase!)
20	19.7	0.71
50	20.2	0.15
100	20.0	0.08

The sharpness exceeds $2/\eta$ at epoch 15, causing a loss increase (catapult). By epoch 20, the catapult has resolved: the network has moved to a region with $\lambda_1 \approx 2/\eta = 20$. It stays near this value for the rest of training — the edge of stability.

39.7 Loss Landscape of Transformer Architectures

The loss landscape of Transformers (attention-based architectures) differs qualitatively from that of MLPs and CNNs due to the softmax attention mechanism and layer normalisation.

Attention as a Lipschitz constraint. The attention function:

$$\text{Attn}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \text{softmax}\left(\frac{\mathbf{Q}\mathbf{K}^\top}{\sqrt{d_k}}\right) \mathbf{V}$$

has Lipschitz constant $L_{\text{Attn}} \leq \|\mathbf{V}\|_2$ (in the ℓ_2 norm on the output). The $1/\sqrt{d_k}$ scaling ensures the input to softmax has unit variance at initialisation (when $\mathbf{Q}, \mathbf{K}$ have variance 1), preventing the softmax from saturating to a one-hot distribution.

Loss landscape smoothness. Dong et al. (2021) show that the Hessian spectrum of a Transformer is more concentrated than that of an MLP: the bulk of eigenvalues is near zero, with a small number of large outlier eigenvalues. This is consistent with the attention mechanism acting as a low-rank projection (each attention head computes a rank-1 update to the residual stream), making the effective curvature low-rank.

Effect of Layer Normalisation. LayerNorm normalises each token's representation:

$$\text{LN}(\mathbf{h}) = \frac{\mathbf{h} - \mu(\mathbf{h})}{\sigma(\mathbf{h})} \odot \square + \square$$

This makes the loss landscape scale-invariant in the hidden dimension: multiplying $\mathbf{W}$ by a constant c does not change the output (since LN will renormalise). As a consequence:

- The Hessian has zero eigenvalues in the “scale direction” of each weight matrix.
- The effective parameter space is the quotient space $\mathbb{R}^d/(\mathbb{R}^+)^L$ (weights modulo per-layer scales).
- Natural gradient in this quotient space is equivalent to weight decay + AdamW (Kosson et al., 2023).

Transformer loss landscape and warm-up. The standard Transformer training recipe requires learning rate warm-up (linearly increase η for the first T_{warmup} steps). The theoretical explanation (Liu et al., 2020): in the early training phase, the adaptive second moment $\hat{v}_t$ in Adam is poorly estimated (it has not yet accumulated enough gradient history). The warm-up prevents large updates from poorly estimated $\hat{v}_t$ from destabilising the loss landscape.

Formally, the update variance from Adam without warm-up is $O(1/\sqrt{1-\beta_2^t})$ for step t . For $\beta_2 = 0.999$ and $t = 1$: variance $\approx 31 \times$ the nominal value. Warm-up reduces η_t by a factor t/T_{warmup} , giving effective initial variance $O(\eta_0/T_{\text{warmup}})$ which is safe.

The loss landscape of a Transformer with L layers and H attention heads has a natural block structure: each attention head contributes an independent rank-1 update, and the MLP blocks contribute higher-rank curvature. The spectrum of the Hessian is well-approximated by a $H \times H$ block structure (one block per head), motivating head-specific learning rates in some optimisation schemes.

40.1 Riemannian Geometry of the Parameter Space

The parameter space $\Theta = \mathbb{R}^d$ can be equipped with a Riemannian metric that reflects the intrinsic structure of the statistical model, leading to much more efficient optimisation than Euclidean gradient descent.

The problem with Euclidean gradients. The ordinary gradient $\nabla_{\square} \mathcal{L}$ measures steepest ascent with respect to the Euclidean metric $\|\delta \square\|_2^2 = \sum_i \delta \theta_i^2$. However, equal Euclidean steps in different parameter directions can produce very different changes in the model's output distribution. A step in a "sensitive" direction (e.g., a small weight in the final layer of a deep network) may produce large output changes, while the same step in an "insensitive" direction (e.g., a weight in a saturated neuron) may produce almost none.

Riemannian Gradient Descent

Given a Riemannian metric $\mathbf{G}(\square)$ (a positive definite matrix at each point), the **natural gradient** is:

$$\tilde{\nabla}_{\square} \mathcal{L} = \mathbf{G}(\square)^{-1} \nabla_{\square} \mathcal{L}$$

The natural gradient update $\square \leftarrow \square - \eta \mathbf{G}^{-1} \nabla \mathcal{L}$ is *steepest descent* under the metric $\mathbf{G}$.

The Fisher information metric. For a parametric family $p_{\square}(y|\mathbf{x})$, the natural Riemannian metric is the Fisher information matrix:

$$\mathbf{F}(\square) = \mathbb{E}_{(\mathbf{x}, y) \sim p_{\square}} [\nabla_{\square} \log p_{\square}(y|\mathbf{x}) \nabla_{\square} \log p_{\square}(y|\mathbf{x})^{\top}]$$

This metric has a deep statistical meaning: it quantifies the sensitivity of the distribution $p_{\square}$ to infinitesimal parameter changes, as measured by the KL divergence:

$$\text{KL}(p_{\square+\delta \square} \| p_{\square}) = \frac{1}{2} \delta \square^{\top} \mathbf{F}(\square) \delta \square + O(\|\delta \square\|^3)$$

Coordinate invariance. The key theoretical property of natural gradient descent is its invariance under reparametrisation. If we transform $\square \rightarrow \square = h(\square)$, the natural gradient update in $\square$ -coordinates produces exactly the same trajectory in function space as the natural gradient update in $\square$ -coordinates. This is not true for ordinary gradient descent.

EXAMPLE — Natural Gradient for a Softmax Classifier

For $p_\theta(y|\mathbf{x}) = \text{softmax}(\mathbf{W}\mathbf{x})_y$, the Fisher matrix is:

$$\mathbf{F} = \mathbb{E}_{\mathbf{x}} [\mathbf{J}_{\mathbf{x}}^\top \text{Cov}(y|\mathbf{x}) \mathbf{J}_{\mathbf{x}}]$$

where $\mathbf{J}_{\mathbf{x}}$ is the Jacobian of the logits with respect to $\square$. For a K -class problem with d parameters, $\mathbf{F} \in \mathbb{R}^{d \times d}$ captures correlations between all parameters.

Connection to second-order methods. When $p_\square(y|\mathbf{x})$ is used as a likelihood model with squared loss, the Fisher matrix approximates the Gauss-Newton matrix:

$$\mathbf{F} \approx \mathbf{J}^\top \mathbf{J}$$

where $\mathbf{J}$ is the Jacobian of the network outputs. The Gauss-Newton update $\square \leftarrow \square - (\mathbf{J}^\top \mathbf{J})^{-1} \mathbf{J}^\top \mathbf{r}$ (where $\mathbf{r}$ is the residual) is the Newton step for the squared loss and converges quadratically near a minimum. Natural gradient descent thus interpolates between gradient descent and Newton's method.

In practice, $\mathbf{F}$ is a $d \times d$ matrix with $d \sim 10^8$, making exact computation and inversion impossible. The key algorithmic challenge is to efficiently approximate $\mathbf{F}^{-1} \nabla \mathcal{L}$. This is addressed by K-FAC (§40.3) and related methods.

A neural network defines a statistical model: a family of distributions $p_\square(\mathbf{y}|\mathbf{x})$. The space of distributions is not Euclidean — it has a natural Riemannian metric, the *Fisher information metric*.

Fisher Information Matrix

For a model $p_\square(\mathbf{y}|\mathbf{x})$, the Fisher information matrix is:

$$\mathbf{F}(\square) = \mathbb{E}_{p_\square} [\nabla_\square \log p_\square \cdot (\nabla_\square \log p_\square)^\top]$$

Equivalently, $F_{ij} = \mathbb{E} \left[\frac{\partial \log p}{\partial \theta_i} \frac{\partial \log p}{\partial \theta_j} \right]$. $\mathbf{F}$ is symmetric positive semi-definite.

Why Fisher, not Euclidean? Two parameter vectors $\square$ and $\square + \delta\square$ at Euclidean distance ϵ may define distributions that are statistically close or far. The Fisher metric measures the KL divergence between nearby distributions:

$$\text{KL}(p_\square \| p_{\square + \delta\square}) \approx \frac{1}{2} \delta\square^\top \mathbf{F}(\square) \delta\square$$

40.2 Natural Gradient Update

The Amari (1998) formulation. Amari's natural gradient descent solves the constrained optimisation:

$$\min_{\delta\square} \nabla_\square \mathcal{L}^\top \delta\square \quad \text{subject to} \quad \delta\square^\top \mathbf{F} \delta\square = \epsilon^2$$

The solution is $\delta\square^* = -\eta \mathbf{F}^{-1} \nabla \mathcal{L}$, giving the natural gradient update:

$$\square_{t+1} = \square_t - \eta \mathbf{F}(\square_t)^{-1} \nabla_{\square_t} \mathcal{L}$$

The constraint uses the Fisher metric $\mathbf{F}$ rather than the Euclidean metric $\mathbf{I}$, so the update moves a fixed distance in the space of distributions rather than in parameter space.

Empirical Fisher vs. true Fisher. In practice, the true Fisher $\mathbf{F}$ is approximated by the *empirical Fisher*:

$$\hat{\mathbf{F}} = \frac{1}{n} \sum_{i=1}^n \nabla_{\square} \log p_{\square}(y_i | \mathbf{x}_i) \nabla_{\square} \log p_{\square}(y_i | \mathbf{x}_i)^{\top}$$

This replaces the expectation over p_{θ} with an average over training data. The empirical Fisher is biased when $p_{\theta} \neq p_{\text{data}}$, but is computationally tractable.

Gauss-Newton vs. Natural Gradient

For squared loss $\mathcal{L} = \frac{1}{2n} \|\mathbf{f}(\square) - \mathbf{y}\|^2$:

- **Gauss-Newton:** $\mathbf{G} = \frac{1}{n} \mathbf{J}^{\top} \mathbf{J}$, update $= -\eta (\mathbf{J}^{\top} \mathbf{J})^{-1} \mathbf{J}^{\top} \mathbf{r}$.
- **Natural gradient:** $\mathbf{F} = \frac{1}{n} \mathbf{J}^{\top} \mathbf{J}$ when $p_{\theta} = \mathcal{N}(f(\square), \sigma^2 \mathbf{I})$.

For Gaussian output models, Gauss-Newton and natural gradient coincide. For non-Gaussian models (e.g., softmax), they differ.

Convergence theory. For a convex loss and full Fisher, the natural gradient update converges at a rate $O(1/t)$ like gradient descent, but with a much smaller constant: the effective condition number is 1 (all directions are equally difficult in Fisher-metric space). This *condition number independence* is the key advantage: while gradient descent can take $O(\kappa)$ iterations to converge for a function with condition number κ , the natural gradient converges in $O(1)$ iterations independent of κ .

Practical implementation. Direct inversion of $\mathbf{F}$ requires $O(d^3)$ operations, prohibitive for large d . Efficient approximations include:

- **Diagonal approximation:** $\mathbf{F} \approx \text{diag}(\mathbf{F})$. Reduces to AdaGrad-style per-parameter learning rates.
- **Block-diagonal:** One block per layer. Ignores inter-layer correlations but is $O(d)$ per layer.
- **K-FAC:** Kronecker-factored block-diagonal. See §40.3.
- **Lanczos approximation:** Iterative eigenvector methods to approximate $\mathbf{F}^{-1}$ in a low-dimensional subspace.

EXAMPLE — Natural Gradient for a Single Linear Layer

For $\mathbf{h} = \mathbf{W}\mathbf{x}$ with $\mathbf{W} \in \mathbb{R}^{m \times n}$, the Fisher (for squared loss) is:

$$\mathbf{F}_W = \mathbb{E}[\mathbf{x}\mathbf{x}^{\top}] \otimes \mathbf{I}_m = \mathbf{A} \otimes \mathbf{I}_m$$

where $\mathbf{A} = \mathbb{E}[\mathbf{x}\mathbf{x}^{\top}]$ is the input covariance. The natural gradient is:

$$\tilde{\nabla}_W \mathcal{L} = \mathbf{F}_W^{-1} \text{vec}(\nabla_W \mathcal{L}) = \text{vec}(\nabla_W \mathcal{L} \mathbf{A}^{-1})$$

or in matrix form: $\tilde{\nabla}_W = (\nabla_W) \mathbf{A}^{-1}$. This *whitens* the input covariance, making the effective loss landscape isotropic.

In the Riemannian manifold, the steepest descent direction with respect to the Fisher metric is the *natural gradient*:

$$\begin{aligned}\tilde{\nabla}_{\square} \mathcal{L} &= \mathbf{F}(\square)^{-1} \nabla_{\square} \mathcal{L} \\ \square_{t+1} &= \square_t - \eta \mathbf{F}(\square_t)^{-1} \nabla \mathcal{L}(\square_t)\end{aligned}$$

Derivation via constrained optimisation. We minimise $\mathcal{L}(\square + \delta\square)$ subject to the KL constraint $\delta\square^\top \mathbf{F} \delta\square \leq \epsilon^2$:

$$\min_{\delta\square} \nabla \mathcal{L}^\top \delta\square \quad \text{s.t.} \quad \delta\square^\top \mathbf{F} \delta\square \leq \epsilon^2$$

Using Lagrangians: $\nabla \mathcal{L} + 2\lambda \mathbf{F} \delta\square = \mathbf{0}$, so $\delta\square = -\frac{1}{2\lambda} \mathbf{F}^{-1} \nabla \mathcal{L}$.

40.3 K-FAC: Kronecker-Factored Approximate Curvature

K-FAC (Martens & Grosse, 2015) is the most practically successful approximation to the natural gradient for deep learning. It exploits the Kronecker structure of the Fisher matrix that arises naturally from the layer-wise computation of a neural network.

Setup. Consider a single fully-connected layer with weight matrix $\mathbf{W} \in \mathbb{R}^{m \times n}$:

$$\mathbf{s}^{(l)} = \mathbf{W} \mathbf{a}^{(l-1)}, \quad \mathbf{a}^{(l)} = \phi(\mathbf{s}^{(l)})$$

The gradient with respect to $\mathbf{W}$ is:

$$\frac{\partial \mathcal{L}}{\partial \mathbf{W}} = \mathbf{g}^{(l)} (\mathbf{a}^{(l-1)})^\top$$

where $\mathbf{g}^{(l)} = \partial \mathcal{L} / \partial \mathbf{s}^{(l)}$ is the pre-activation gradient. Vectorising: $\text{vec}(\partial \mathcal{L} / \partial \mathbf{W}) = \mathbf{a}^{(l-1)} \otimes \mathbf{g}^{(l)}$.

The Fisher block for layer l .

$$\mathbf{F}_l = \mathbb{E}[(\mathbf{a} \otimes \mathbf{g})(\mathbf{a} \otimes \mathbf{g})^\top] = \mathbb{E}[(\mathbf{a} \mathbf{a}^\top) \otimes (\mathbf{g} \mathbf{g}^\top)]$$

The key K-FAC approximation is to assume $\mathbf{a}$ and $\mathbf{g}$ are statistically independent:

$$\mathbf{F}_l \approx \mathbb{E}[\mathbf{a} \mathbf{a}^\top] \otimes \mathbb{E}[\mathbf{g} \mathbf{g}^\top] = \mathbf{A} \otimes \mathbf{G}$$

where $\mathbf{A} = \mathbb{E}[\mathbf{a} \mathbf{a}^\top] \in \mathbb{R}^{n \times n}$ and $\mathbf{G} = \mathbb{E}[\mathbf{g} \mathbf{g}^\top] \in \mathbb{R}^{m \times m}$.

Efficient inversion. Using the Kronecker product identity $(\mathbf{A} \otimes \mathbf{G})^{-1} = \mathbf{A}^{-1} \otimes \mathbf{G}^{-1}$:

$$\mathbf{F}_l^{-1} \text{vec}(\nabla_{\mathbf{W}} \mathcal{L}) = (\mathbf{A}^{-1} \otimes \mathbf{G}^{-1}) \text{vec}(\nabla_{\mathbf{W}} \mathcal{L})$$

In matrix form, the natural gradient update for $\mathbf{W}$ is:

$$\nabla_{\mathbf{W}} \mathcal{L} = \mathbf{G}^{-1} (\nabla_{\mathbf{W}} \mathcal{L}) \mathbf{A}^{-1}$$

Cost reduction. Direct inversion of $\mathbf{F}_l \in \mathbb{R}^{mn \times mn}$ costs $O((mn)^3)$. K-FAC requires inverting $\mathbf{A} \in \mathbb{R}^{n \times n}$ at cost $O(n^3)$ and $\mathbf{G} \in \mathbb{R}^{m \times m}$ at cost $O(m^3)$, reducing total inversion cost from $O((mn)^3)$ to $O(n^3 + m^3)$. For $m = n = 1000$: from 10^{18} to 2×10^9 operations — nine orders of magnitude.

EXAMPLE — K-FAC Update for a 2-Layer MLP

Layer 1: $\mathbf{W}_1 \in \mathbb{R}^{512 \times 100}$, activations $\mathbf{a} \in \mathbb{R}^{100}$, pre-gradients $\mathbf{g} \in \mathbb{R}^{512}$.

Running statistics (exponential moving average):

$$\begin{aligned}\hat{\mathbf{A}} &= \beta \hat{\mathbf{A}} + (1 - \beta) \mathbf{a} \mathbf{a}^\top \in \mathbb{R}^{100 \times 100} \\ \hat{\mathbf{G}} &= \beta \hat{\mathbf{G}} + (1 - \beta) \mathbf{g} \mathbf{g}^\top \in \mathbb{R}^{512 \times 512}\end{aligned}$$

Damped inversion: $\tilde{\mathbf{A}} = \hat{\mathbf{A}} + \pi \mathbf{I}$ (Tikhonov regularisation, π set to match the Frobenius norms of $\hat{\mathbf{A}}$ and $\hat{\mathbf{G}}$).

Natural gradient: $\nabla_{\mathbf{w}_1} \mathcal{L} = \hat{\mathbf{G}}^{-1} (\nabla_{\mathbf{w}_1} \mathcal{L}) \hat{\mathbf{A}}^{-1}$.

Direct inversion cost: $100^3 + 512^3 = 1.35 \times 10^8$ vs. naive $(100 \cdot 512)^3 = 3.3 \times 10^{17}$ — speedup of $2.4 \times 10^9 \times$.

K-FAC vs. Adam. K-FAC uses the full layer-wise curvature structure and converges to lower loss in fewer iterations than Adam for ResNets and Transformers. However, it requires maintaining and inverting $\mathbf{A}$ and $\mathbf{G}$ matrices for every layer, adding memory overhead proportional to $\sum_l (m_l^2 + n_l^2)$. For a 70B parameter LLM, this overhead is prohibitive. K-FAC is most practical for medium-scale networks (up to $\sim 10^8$ parameters) where the memory overhead is acceptable.

Computing $\mathbf{F}^{-1}$ exactly requires $O(d^2)$ storage and $O(d^3)$ inversion — infeasible for large networks. K-FAC approximates $\mathbf{F}$ as a block-diagonal matrix, one block per layer, where each block is a Kronecker product:

$$\mathbf{F}_l \approx \hat{\mathbf{A}}_{l-1} \otimes \hat{\mathbf{G}}_l$$

where $\hat{\mathbf{A}}_{l-1} = \mathbb{E}[\mathbf{a}_{l-1} \mathbf{a}_{l-1}^\top]$ (input covariance) and $\hat{\mathbf{G}}_l = \mathbb{E}[\mathbf{g}_l \mathbf{g}_l^\top]$ (gradient covariance).

Inversion. Using the Kronecker product identity $(A \otimes B)^{-1} = A^{-1} \otimes B^{-1}$, inverting the d_l^2 -dimensional block reduces to two small matrix inversions.

EXAMPLE — K-FAC for a linear layer with 3 inputs, 2 outputs

Layer: $\mathbf{z}_l = \mathbf{W}_l \mathbf{a}_{l-1}$. If $\mathbf{a}_{l-1} \in \mathbb{R}^3$, $\mathbf{W}_l \in \mathbb{R}^{2 \times 3}$.

$\hat{\mathbf{A}}_{l-1} \in \mathbb{R}^{3 \times 3}$ (input covariance), $\hat{\mathbf{G}}_l \in \mathbb{R}^{2 \times 2}$ (gradient covariance). $\mathbf{F}_l \approx \hat{\mathbf{A}}_{l-1} \otimes \hat{\mathbf{G}}_l \in \mathbb{R}^{6 \times 6}$.

Inversion: $\mathbf{F}_l^{-1} \approx \hat{\mathbf{A}}_{l-1}^{-1} \otimes \hat{\mathbf{G}}_l^{-1}$. Cost: invert a 3×3 and a 2×2 — trivial.

40.4 Quasi-Newton Methods and L-BFGS

Quasi-Newton methods approximate the inverse Hessian iteratively from gradient differences, achieving superlinear convergence without the $O(d^3)$ cost of full Newton steps. L-BFGS (Limited-memory BFGS) is the most successful second-order method for large-scale unconstrained optimisation.

The secant condition. The Hessian approximation $\mathbf{B}_t \approx \nabla^2 \mathcal{L}(\square_t)$ must satisfy the secant condition:

$$\mathbf{B}_{t+1} \mathbf{s}_t = \mathbf{y}_t$$

where $\mathbf{s}_t = \square_{t+1} - \square_t$ (step vector) and $\mathbf{y}_t = \nabla \mathcal{L}(\square_{t+1}) - \nabla \mathcal{L}(\square_t)$ (gradient change). The secant condition ensures $\mathbf{B}_{t+1}$ has the “right” curvature in the direction $\mathbf{s}_t$.

BFGS update. The BFGS (Broyden-Fletcher-Goldfarb-Shanno) formula updates the inverse Hessian approximation $\mathbf{H}_t = \mathbf{B}_t^{-1}$:

$$\mathbf{H}_{t+1} = (\mathbf{I} - \rho_t \mathbf{s}_t \mathbf{y}_t^\top) \mathbf{H}_t (\mathbf{I} - \rho_t \mathbf{y}_t \mathbf{s}_t^\top) + \rho_t \mathbf{s}_t \mathbf{s}_t^\top$$

where $\rho_t = 1/(\mathbf{y}_t^\top \mathbf{s}_t)$. If $\rho_t > 0$ (Wolfe condition), $\mathbf{H}_{t+1}$ is symmetric positive definite.

L-BFGS: the limited-memory variant. Storing the full $\mathbf{H}_t \in \mathbb{R}^{d \times d}$ requires $O(d^2)$ memory — infeasible for large d . L-BFGS stores only the last m curvature pairs $\{(\mathbf{s}_k, \mathbf{y}_k)\}_{k=t-m}^t$ and computes $\mathbf{H}_t^{-1} \mathbf{g}$ implicitly via the two-loop recursion:

L-BFGS Two-Loop Recursion

To compute $\mathbf{p} = \mathbf{H}_t \mathbf{g}$ using m stored curvature pairs:

1. $\mathbf{q} \leftarrow \mathbf{g}$
2. **Backward loop** $k = t-1, t-2, \dots, t-m$:
 $\alpha_k \leftarrow \rho_k \mathbf{s}_k^\top \mathbf{q}$;
 $\mathbf{q} \leftarrow \mathbf{q} - \alpha_k \mathbf{y}_k$
3. $\mathbf{p} \leftarrow \mathbf{H}_0 \mathbf{q}$ (initial Hessian, typically $\gamma_t \mathbf{I}$ with $\gamma_t = \mathbf{s}_{t-1}^\top \mathbf{y}_{t-1} / \|\mathbf{y}_{t-1}\|^2$)
4. **Forward loop** $k = t-m, \dots, t-2, t-1$:
 $\beta_k \leftarrow \rho_k \mathbf{y}_k^\top \mathbf{p}$;
 $\mathbf{p} \leftarrow \mathbf{p} + (\alpha_k - \beta_k) \mathbf{s}_k$
5. **Output:** $\mathbf{p} = \mathbf{H}_t \mathbf{g}$ (the L-BFGS direction)

Memory: $O(md)$. **Cost per step:** $O(md)$ (2 passes over m vectors).

Line search and Wolfe conditions. Unlike gradient descent with fixed step size, L-BFGS requires an approximate line search to satisfy the Wolfe conditions at each step:

$$\begin{aligned} \mathcal{L}(\square + \alpha \mathbf{p}) &\leq \mathcal{L}(\square) + c_1 \alpha \nabla \mathcal{L}^\top \mathbf{p} \quad (\text{sufficient decrease}) \\ |\nabla \mathcal{L}(\square + \alpha \mathbf{p})^\top \mathbf{p}| &\leq c_2 |\nabla \mathcal{L}(\square)^\top \mathbf{p}| \quad (\text{curvature condition}) \end{aligned}$$

with $0 < c_1 < c_2 < 1$ (typically $c_1 = 10^{-4}$, $c_2 = 0.9$). The curvature condition ensures the curvature pair $(\mathbf{s}_t, \mathbf{y}_t)$ has $\mathbf{s}_t^\top \mathbf{y}_t > 0$, guaranteeing the BFGS update produces a positive-definite $\mathbf{H}_{t+1}$.

Convergence. L-BFGS converges superlinearly near a local minimum:

$$\|\square_{t+1} - \square^*\| \leq C_t \|\square_t - \square^*\|^{1+\gamma}$$

for some $\gamma \in (0, 1]$ that depends on how well the stored curvature captures the true Hessian. In practice, $m = 5\text{--}20$ curvature pairs typically achieves convergence in $O(\text{condition_number}^{0.5})$ iterations — much faster than gradient descent's $O(\text{condition_number})$.

EXAMPLE — L-BFGS vs. Adam on a Convex Problem

Logistic regression on MNIST: $d = 784 \times 10 = 7840$ parameters, $n = 60000$.

True Hessian condition number: $\kappa \approx 10^4$ (varies by regularisation).

Optimiser	Steps to $\ \nabla \mathcal{L}\ \leq 10^{-4}$	Final loss
SGD ($\eta = 0.01$)	5000	0.218
Adam ($\eta = 10^{-3}$)	1200	0.217
L-BFGS ($m = 10$)	120	0.214

L-BFGS converges 10× faster than Adam in terms of gradient evaluations. However, L-BFGS requires full-batch gradients (stochastic L-BFGS is an active research area); for online/mini-batch settings, Adam remains dominant.

Full-batch L-BFGS is used in PyTorch for tasks where the full dataset fits in memory: feature extraction with fixed embeddings, meta-learning, and fine-tuning the last layer of large models.

Stochastic L-BFGS. Mini-batch curvature pairs $(\mathbf{s}_t, \mathbf{y}_t)$ may have $\mathbf{s}_t^\top \mathbf{y}_t \leq 0$ due to gradient noise, violating the Wolfe condition. Remedies:

- *Curvature pair rejection*: discard pairs with $\mathbf{s}_t^\top \mathbf{y}_t \leq \epsilon \|\mathbf{s}_t\| \|\mathbf{y}_t\|$.
- *Damped BFGS*: modify $\mathbf{y}_t \leftarrow \mathbf{y}_t + \mu \mathbf{s}_t$ with $\mu > 0$ to guarantee positive definiteness.
- *SQN (Stochastic Quasi-Newton)*: use a larger mini-batch for curvature estimation than for gradient estimation.

40.5 Shampoo and Distributed Second-Order Optimisation

Shampoo (Gupta et al., 2018; Anil et al., 2020) extends the natural gradient idea to practical large-scale training by maintaining Kronecker-factored preconditioners for each weight tensor.

Tensor gradient structure. For a weight matrix $\mathbf{W} \in \mathbb{R}^{m \times n}$ with gradient $\mathbf{G} = \nabla_{\mathbf{W}} \mathcal{L} \in \mathbb{R}^{m \times n}$, the full second-order preconditioner would require $\mathbf{F} \in \mathbb{R}^{mn \times mn}$. Shampoo approximates:

$$\mathbf{F} \approx \mathbf{L} \otimes \mathbf{R}, \quad \mathbf{L} = \mathbb{E}[\mathbf{G}\mathbf{G}^\top] \in \mathbb{R}^{m \times m}, \quad \mathbf{R} = \mathbb{E}[\mathbf{G}^\top \mathbf{G}] \in \mathbb{R}^{n \times n}$$

The Shampoo update is:

$$\mathbf{W}_{t+1} = \mathbf{W}_t - \eta \mathbf{L}_t^{-1/4} \mathbf{G}_t \mathbf{R}_t^{-1/4}$$

where $\mathbf{L}_t = \sum_{s \leq t} \mathbf{G}_s \mathbf{G}_s^\top$ and $\mathbf{R}_t = \sum_{s \leq t} \mathbf{G}_s^\top \mathbf{G}_s$ are accumulated statistics. The $-1/4$ power (rather than $-1/2$) accounts for the Kronecker product: $(\mathbf{L} \otimes \mathbf{R})^{-1/2} = \mathbf{L}^{-1/2} \otimes \mathbf{R}^{-1/2}$ has the same effect as $\mathbf{L}^{-1/4} \cdot \mathbf{R}^{-1/4}$ applied to the matrix gradient.

Matrix power via eigendecomposition. Computing $\mathbf{L}^{-1/4}$ requires the eigendecomposition $\mathbf{L} = \mathbf{V} \mathbf{\Lambda} \mathbf{V}^\top$:

$$\mathbf{L}^{-1/4} = \mathbf{V} \mathbf{\Lambda}^{-1/4} \mathbf{V}^\top$$

This is $O(m^3)$ per step, feasible for layer sizes $m \leq 1000$. For larger layers, Shampoo is applied per-block (blocking the weight matrix into sub-matrices). In distributed training, each device maintains and inverts its own preconditioner shards.

Convergence guarantee. For convex objectives:

$$\sum_{t=1}^T \mathcal{L}(\mathbf{W}_t) - T\mathcal{L}^* \leq O(\|\mathbf{W}_0 - \mathbf{W}^*\|_F \sqrt{T}) \cdot \text{tr}(\mathbf{L}_T)^{1/4} \text{tr}(\mathbf{R}_T)^{1/4}$$

The regret bound is tighter than AdaGrad by a factor $\sqrt{mn/(m+n)}$ in the worst case, since Shampoo exploits the matrix structure of the gradient.

EXAMPLE — Shampoo vs. Adam on Transformer Language Model

GPT-style model, 1B parameters, trained on 100B tokens:

Optimiser	Validation perplexity	Steps to PPL = 20
Adam	18.4	95 000
Distributed Shampoo	17.1	71 000

Shampoo converges 25% faster in steps to the same perplexity, at a $\sim 2\times$ computational overhead per step. For large-scale training budgets, the step reduction often outweighs the per-step cost. Google’s production LLM training (PaLM, Gemini) reportedly uses variants of distributed Shampoo.

Sketched Newton methods. An alternative to Kronecker factorisation is sketch-and-project Newton: randomly project the Hessian to a low-dimensional subspace $\mathbf{S} \in \mathbb{R}^{d \times k}$ ($k \ll d$), compute the exact Newton step in the sketch, and project back. The sketched Newton update:

$$\theta_{t+1} = \theta_t - (\mathbf{S}^\top \mathbf{H}_t \mathbf{S})^{-1} \mathbf{S}^\top \nabla \mathcal{L}_t$$

has $O(kd)$ per-step cost and achieves linear convergence with rate $1 - k/d$ in the eigendirections captured by the sketch. For Gaussian sketches $\mathbf{S}$, the expected rate is $1 - k/\text{rank}(\mathbf{H}_t)$, near-optimal when $k \approx \text{rank}(\mathbf{H}_t)$.

PAC-Bayes and Generalisation Bounds

We derive PAC-Bayes bounds, which give non-vacuous guarantees on the test loss of stochastic classifiers in terms of KL divergence between a posterior and a prior over the hypothesis space.

41.1 The Generalisation Problem

The central question of statistical learning theory: why do deep networks trained on finite data generalise to unseen examples? Classical results (VC theory, Rademacher complexity) give pessimistic bounds that vacuously predict that overparameterised networks should overfit. Understanding why they do not is one of the deepest open problems in machine learning.

Setup. Let $\mathcal{D} = \{(\mathbf{x}_i, y_i)\}_{i=1}^n$ be training data drawn i.i.d. from an unknown distribution $\mathcal{P}$. For a hypothesis $h \in \mathcal{H}$:

- **Training risk:** $\hat{R}(h) = \frac{1}{n} \sum_{i=1}^n \ell(h(\mathbf{x}_i), y_i)$
- **True risk:** $R(h) = \mathbb{E}_{(\mathbf{x}, y) \sim \mathcal{P}}[\ell(h(\mathbf{x}), y)]$
- **Generalisation gap:** $R(h) - \hat{R}(h)$

The classical VC bound. For a binary classifier with VC dimension d_{VC} :

$$R(h) \leq \hat{R}(h) + O\left(\sqrt{\frac{d_{\text{VC}} \log(n/d_{\text{VC}})}{n}}\right)$$

For modern networks, $d_{\text{VC}} \sim d$ (number of parameters), giving bounds of order $\sqrt{d/n} \gg 1$ — vacuous.

Why classical bounds fail. Modern networks are vastly overparameterised: $d \gg n$. Yet they generalise well. Classical uniform convergence bounds assume worst-case hypothesis, but SGD implicitly selects a specific (and well-generalising) hypothesis via its implicit regularisation. Tighter bounds must account for the algorithm, not just the hypothesis class.

Benign overfitting. A striking recent result (Bartlett et al., 2020): for linear regression with overparameterised models, the minimum-norm interpolating solution can simultaneously have zero training error and optimal test error — “overfitting” does not preclude generalisation. The condition is that the data covariance has sufficiently many small eigenvalues (effective rank condition).

The double descent phenomenon. Classical statistics predicts a U-shaped bias-variance tradeoff: test error decreases as model capacity increases to the bias-limited regime, then increases (overfitting) as variance grows. Double descent (Belkin et al., 2019) shows a second descent beyond the interpolation threshold:

Regime	Behaviour
Underparameterised ($d < n$)	Classical U-shape: error decreases then rises.
Interpolation threshold ($d = n$)	Test error peaks (the “double descent peak”).
Overparameterised ($d \gg n$)	Test error decreases again with more parameters.

Mechanism. At the interpolation threshold $d \approx n$, the model is barely able to fit the training data; the fitting is “rigid” and sensitive to noise. Above the threshold, there are many solutions fitting the training data; the minimum-norm solution (selected by SGD or gradient flow) is “smooth” and generalises better than the noisy interpolating solutions near the threshold.

EXAMPLE — Double Descent for Linear Regression

$n = 1000$, features $\mathbf{x}_i \sim \mathcal{N}(0, \mathbf{I}_d)$, labels $y_i = \mathbf{w}^{*\top} \mathbf{x}_i + \varepsilon_i$, $\varepsilon_i \sim \mathcal{N}(0, 0.5)$.

Estimator: minimum-norm least squares $\hat{\mathbf{w}} = \mathbf{X}^\dagger \mathbf{y}$.

- $d = 100$ ($d/n = 0.1$): test MSE ≈ 0.55 (slight overfitting).
- $d = 900$ ($d/n = 0.9$): test MSE ≈ 0.80 (near threshold, high variance).
- $d = 1000$ ($d/n = 1.0$): test MSE ≈ 2.5 (threshold, max error).
- $d = 5000$ ($d/n = 5$): test MSE ≈ 0.52 (second descent).
- $d = 50000$ ($d/n = 50$): test MSE ≈ 0.51 (close to Bayes optimal 0.5).

The test error at $d/n = 50$ is lower than at $d/n = 0.1$, despite using $500\times$ more parameters.

Generalisation is the central challenge of machine learning: a model trained on a finite sample $\mathcal{D} = \{(\mathbf{x}_i, y_i)\}_{i=1}^n$ must perform well on unseen data drawn from the same distribution $\mathcal{P}$.

Formal setup. Let $h : \mathcal{X} \rightarrow \mathcal{Y}$ be a hypothesis, $\ell : \mathcal{Y} \times \mathcal{Y} \rightarrow \mathbb{R}_+$ a loss function. The *population risk* (true error) and *empirical risk* (training error) are:

$$R(h) = \mathbb{E}_{(\mathbf{x}, y) \sim \mathcal{P}}[\ell(h(\mathbf{x}), y)], \quad \hat{R}(h) = \frac{1}{n} \sum_{i=1}^n \ell(h(\mathbf{x}_i), y_i)$$

The *generalisation gap* is $R(h) - \hat{R}(h)$. The goal is to minimise $R(h)$ while only having access to $\hat{R}(h)$.

Overfitting and Underfitting

- **Overfitting:** $\hat{R}(h) \ll R(h)$. The model fits noise in the training data.
- **Underfitting:** $\hat{R}(h) \approx R(h)$ but both are large. The model lacks capacity.
- **Good generalisation:** $\hat{R}(h) \approx R(h) \approx R(h^*)$ where $h^* = \arg \min_h R(h)$.

Classical approaches: VC theory. Vapnik-Chervonenkis theory provides uniform convergence bounds. For a hypothesis class $\mathcal{H}$ with VC dimension d_{VC} :

$$P\left(\sup_{h \in \mathcal{H}} |R(h) - \hat{R}(h)| > \epsilon\right) \leq 4 m_{\mathcal{H}}(2n) e^{-n\epsilon^2/8}$$

where $m_{\mathcal{H}}(n)$ is the growth function. For $m_{\mathcal{H}}(n) \leq (en/d_{\text{VC}})^{d_{\text{VC}}}$, the bound becomes:

$$R(h) \leq \hat{R}(h) + O\left(\sqrt{\frac{d_{\text{VC}} \log(n/d_{\text{VC}}) + \log(1/\delta)}{n}}\right)$$

Why VC theory fails for deep learning. Modern neural networks have $d_{\text{VC}} = \Theta(d \log d)$ where $d \sim 10^8$. With $n \sim 10^6$ training examples, the VC bound is vacuous (> 1). Yet

these networks generalise remarkably well. This *generalisation puzzle* has driven a decade of theoretical research.

The interpolation regime. Belkin et al. (2019) showed that models which exactly interpolate the training data (zero training error) can still generalise well if they are sufficiently *smooth*. This is captured by the double descent phenomenon: as model size increases past interpolation threshold, the test error decreases again. Classical bias-variance tradeoff — which predicts an optimal model size — holds only in the classical (underparameterised) regime.

Rademacher complexity. A sharper alternative to VC theory is Rademacher complexity, which measures the ability of a hypothesis class to fit random labels.

Rademacher Complexity

For a function class $\mathcal{F}$ and dataset $S = (\mathbf{x}_1, \dots, \mathbf{x}_n)$, the empirical Rademacher complexity is:

$$\hat{\mathfrak{R}}_n(\mathcal{F}) = \mathbb{E}_{\square} \left[\sup_{f \in \mathcal{F}} \frac{1}{n} \sum_{i=1}^n \sigma_i f(\mathbf{x}_i) \right]$$

where $\sigma_i \in \{-1, +1\}$ are i.i.d. Rademacher random variables. The population Rademacher complexity is $\mathfrak{R}_n(\mathcal{F}) = \mathbb{E}_S \hat{\mathfrak{R}}_n(\mathcal{F})$.

Rademacher generalisation bound. For any $f \in \mathcal{F}$ and $\delta \in (0, 1)$, with probability $\geq 1 - \delta$:

$$R(f) \leq \hat{R}(f) + 2\hat{\mathfrak{R}}_n(\mathcal{F}) + \sqrt{\frac{\ln(1/\delta)}{2n}}$$

The advantage over VC theory: Rademacher complexity depends on both the function class *and* the data distribution. For neural networks, it captures the effective capacity used by the training algorithm, not the worst-case capacity.

Norm-based Rademacher bounds. For a neural network with L layers, ReLU activations, and weight matrices $\mathbf{W}^{(1)}, \dots, \mathbf{W}^{(L)}$, Golowich et al. (2018) prove:

$$\mathfrak{R}_n(\mathcal{F}) \leq \frac{(\sqrt{2L \ln 2})^{L-1}}{n} \prod_{l=1}^L \|\mathbf{W}^{(l)}\|_F$$

The bound depends on the Frobenius norm product of all weight matrices — it is data-independent. Better bounds use the spectral norm $\|\mathbf{W}\|_2$ (Bartlett et al., 2017):

$$\mathfrak{R}_n(\mathcal{F}) \leq \frac{B \cdot \prod_l \|\mathbf{W}^{(l)}\|_2 \cdot \left(\sum_l \|\mathbf{W}^{(l)}\|_{2,1}^{2/3} \right)^{3/2}}{\sqrt{n}}$$

where $\|\mathbf{W}\|_{2,1} = \max_j \|\mathbf{w}_j\|_2$ and $B = \sup_{\mathbf{x}} \|\mathbf{x}\|_2$ is the input norm bound.

Margin theory. For a classifier with margin γ (minimum distance from the decision boundary in the feature space), the generalisation bound improves:

$$R(f) \leq \hat{R}_\gamma(f) + O\left(\frac{\text{complexity}}{\gamma\sqrt{n}}\right)$$

where $\hat{R}_\gamma$ is the margin loss (fraction of training examples within margin γ). Large-margin classifiers (SVMs, neural networks with large-margin training) generalise better. Modern networks typically have very large margins, partially explaining their good generalisation despite high VC dimension.

EXAMPLE — Effective Sample Size and Generalisation

ResNet-50 on ImageNet: $d \approx 25 \times 10^6$ parameters, $n = 1.28 \times 10^6$ training examples. VC bound: $O(\sqrt{d \log d/n}) = O(\sqrt{25 \times 10^6 \times 18/1.28 \times 10^6}) \approx O(\sqrt{350}) \approx 19$ — vacuous.

Rademacher with spectral norms: product of per-layer spectral norms $\approx 10^3$ for a well-regularised ResNet (each layer has $\|\mathbf{W}\|_2 \approx 3$). Bound $\approx 10^3/\sqrt{1.28 \times 10^6} \approx 0.88$ — marginally vacuous but $20\times$ better than VC.

PAC-Bayes (Dziugaite & Roy, 2017): optimised bound ≈ 0.16 , non-vacuous. The PAC-Bayes bound tightens because it accounts for the specific learned weights, not just the function class.

The correct framework for understanding deep learning generalisation is an active research frontier. Leading theories include: (1) implicit bias of gradient descent toward minimum-norm solutions, (2) PAC-Bayes bounds with data-dependent priors, (3) neural tangent kernel analysis in the infinite-width limit, and (4) algorithmic stability bounds. None fully explains all observed generalisation phenomena.

Let $\mathcal{D} = \{(\mathbf{x}_i, y_i)\}_{i=1}^n$ be a training set drawn i.i.d. from a distribution $\mathcal{P}$. The empirical risk is:

$$\hat{R}(h) = \frac{1}{n} \sum_{i=1}^n \ell(h(\mathbf{x}_i), y_i)$$

The true (population) risk is $R(h) = \mathbb{E}_{(\mathbf{x}, y) \sim \mathcal{P}}[\ell(h(\mathbf{x}), y)]$. Generalisation asks: how large can $R(h) - \hat{R}(h)$ be?

41.2 PAC-Bayes Framework

PAC-Bayes theory (McAllester 1999, Seeger 2002) provides generalisation bounds for *randomised* classifiers (stochastic posteriors over hypotheses). It gives the tightest known bounds for neural networks and is the basis for current non-vacuous guarantees.

Setup. Let P (the prior) be a distribution over hypotheses chosen *before* seeing training data, and Q (the posterior) be chosen after seeing $\mathcal{D}$. Define:

$$R(Q) = \mathbb{E}_{h \sim Q}[R(h)], \quad \hat{R}(Q) = \mathbb{E}_{h \sim Q}[\hat{R}(h)]$$

McAllester PAC-Bayes Bound

For any prior P on $\mathcal{H}$, any $\delta \in (0, 1)$, with probability $\geq 1 - \delta$ over draws of $\mathcal{D}$ of size n , for all posteriors Q simultaneously:

$$R(Q) \leq \hat{R}(Q) + \sqrt{\frac{\text{KL}(Q\|P) + \ln(n/\delta)}{2n}}$$

Proof sketch. The key tool is Donsker-Varadhan's variational formula: for any measurable $\phi : \mathcal{H} \rightarrow \mathbb{R}$:

$$\ln \mathbb{E}_{h \sim P}[e^{\phi(h)}] = \sup_Q \{ \mathbb{E}_{h \sim Q}[\phi(h)] - \text{KL}(Q\|P) \}$$

Setting $\phi(h) = n(R(h) - \hat{R}(h))$ and applying Hoeffding's inequality to control $\mathbb{E}_P[e^\phi]$, then

rearranging:

$$\Pr \left[\exists Q : R(Q) - \hat{R}(Q) > \sqrt{\frac{\text{KL}(Q\|P) + \ln(1/\delta)}{2n}} \right] \leq \delta$$

The bound holds *uniformly* over all posteriors Q , including those chosen after seeing the data.

Interpretation. The bound has two terms:

- $\hat{R}(Q)$: empirical risk of the stochastic classifier.
- $\sqrt{\text{KL}(Q\|P)/(2n)}$: complexity penalty. $\text{KL}(Q\|P)$ measures how much the posterior departs from the prior. If training barely changes Q from P (low KL), the network has memorised little and generalises well.

EXAMPLE — Non-Vacuous PAC-Bayes Bound for a Neural Network

Dziugaite & Roy (2017) minimise the PAC-Bayes bound directly. For a 2-layer MLP (600 hidden units) on MNIST with $n = 55000$:

- Prior: $P = \mathcal{N}(\square_0, \sigma_0^2 \mathbf{I})$ initialised at $\square_0$.
- Posterior: $Q = \mathcal{N}(\square, \sigma^2 \mathbf{I})$ with $\square$ near the trained weights.
- Achieved: $\hat{R}(Q) = 1.9\%$, $\text{KL}(Q\|P) \approx 10^5$, bound = 16.3%.

This is the first non-vacuous generalisation bound for a network trained on real data. Compare: VC bound gives $> 100\%$ (vacuous). The key: $\text{KL}(Q\|P) \approx 10^5$ while $n \approx 5.5 \times 10^4$, so $\sqrt{\text{KL}/(2n)} \approx \sqrt{0.9} \approx 0.95$, giving $\approx 95\%$ — still quite loose but provably meaningful.

PAC-Bayes Setup

- Fix a **prior** P over hypotheses, chosen *before* seeing the data.
- After training, choose a **posterior** Q over hypotheses.
- The stochastic classifier draws $h \sim Q$ at test time.
- $\hat{R}(Q) = \mathbb{E}_{h \sim Q}[\hat{R}(h)]$, $R(Q) = \mathbb{E}_{h \sim Q}[R(h)]$.

McAllester's PAC-Bayes Bound. With probability $\geq 1 - \delta$ over the draw of $\mathcal{D}$:

$$R(Q) \leq \hat{R}(Q) + \sqrt{\frac{\text{KL}(Q\|P) + \ln(n/\delta)}{2(n-1)}}$$

Proof sketch.

Step 1. For any single hypothesis h , Hoeffding's inequality gives:

$$P(R(h) - \hat{R}(h) > \epsilon) \leq e^{-2n\epsilon^2}$$

Step 2. For a distribution Q over hypotheses, the change-of-measure inequality states:

$$\mathbb{E}_{h \sim Q}[f(h)] \leq \text{KL}(Q\|P) + \ln \mathbb{E}_{h \sim P}[e^{f(h)}]$$

Step 3. Apply with $f(h) = 2(n-1)(R(h) - \hat{R}(h))^2$ (after standard symmetrisation), integrate over $\mathcal{D}$, and solve for the bound on $R(Q)$. $\square$

EXAMPLE — PAC-Bayes bound for a Gaussian posterior

Take $P = \mathcal{N}(\mathbf{0}, \sigma^2 \mathbf{I}_d)$ and $Q = \mathcal{N}(\square, \sigma^2 \mathbf{I}_d)$ (same variance, shifted mean).

$$\text{KL}(Q \| P) = \frac{\|\square\|^2}{2\sigma^2}$$

For $\|\square\| = 1, \sigma = 0.1, d = 10^6$: $\text{KL} = 50$.

With $n = 10^5, \delta = 0.05$:

$$R(Q) \leq \hat{R}(Q) + \sqrt{\frac{50 + \ln(2 \times 10^6)}{2 \times (10^5 - 1)}} \approx \hat{R}(Q) + 0.017$$

A non-vacuous (less than 1) bound — remarkable for 10^6 parameters.

41.3 Connection to Information Theory

The PAC-Bayes bound connects generalisation to information theory through the KL divergence. But there are deeper connections: the generalisation gap is fundamentally an information-theoretic quantity.

Mutual information generalisation bound. Russo & Zou (2016) and Xu & Raginsky (2017) prove: for any learning algorithm A (mapping dataset S to hypothesis $h = A(S)$) and bounded loss $\ell \in [0, 1]$:

$$\mathbb{E}[R(A(S))] - \mathbb{E}[\hat{R}(A(S))] \leq \sqrt{\frac{I(A(S); S)}{2n}}$$

where $I(A(S); S)$ is the mutual information between the algorithm's output and the training set.

Interpretation. If the algorithm's output is nearly independent of the training data ($I(A(S); S) \approx 0$), the generalisation gap is small. SGD generalises well partly because its noisy updates make the final hypothesis depend weakly on any individual training point.

Per-example bound. A tighter version uses per-example information:

$$\mathbb{E}[R(A(S))] - \mathbb{E}[\hat{R}(A(S))] \leq \frac{1}{n} \sum_{i=1}^n \sqrt{\frac{I(A(S); z_i)}{2}}$$

where $z_i = (\mathbf{x}_i, y_i)$ is the i -th training point. This rewards algorithms that learn from each example only weakly — consistent with the DP-SGD framework (§45).

EXAMPLE — Mutual Information Bound for a Gaussian Model

Linear regression, $n = 1000$ examples, $d = 100$ features, Algorithm A = ordinary least squares.

$A(S) = (\mathbf{X}^\top \mathbf{X})^{-1} \mathbf{X}^\top \mathbf{y}$ depends on all training points. The mutual information $I(A(S); S) = \frac{d}{2} \ln(1 + n \|\square^*\|^2 \sigma^{-2})$ (Gaussian formula).

For $\|\square^*\| = 1, \sigma = 1$: $I = 50 \ln(1001) \approx 345$ bits.

MI bound: $\sqrt{345/(2 \times 1000)} = 0.41$. Actual generalisation gap $\approx d/n = 0.1$ (from exact analysis). The MI bound is loose by $4\times$ but is tight in the dependence on d and n .

Tightening via CMI. Steinke & Zakynthinou (2020) give a tighter “conditional mutual information” (CMI) bound using a ghost dataset, achieving bounds that scale as $O(\sqrt{I_{\text{CMI}}/n})$ where $I_{\text{CMI}} \leq I(A(S); S)$ is often much smaller.

Information theory provides some of the sharpest generalisation bounds via the *input-output mutual information* framework of Russo & Zou (2016) and Bu et al. (2020).

The mutual information bound. Let $\mathcal{A}$ be a randomised learning algorithm that maps a dataset $S \in \mathcal{Z}^n$ to a hypothesis $W \in \mathcal{W}$. Let $I(W; S)$ denote the mutual information between the output hypothesis and the training set. Then:

$$\mathbb{E}_{W,S}[R(W) - \hat{R}(W)] \leq \sqrt{\frac{2\sigma^2 I(W; S)}{n}}$$

where the loss is σ^2 -sub-Gaussian. This bound is *algorithm-dependent* and can be tight: algorithms that “remember” little about the training data (small $I(W; S)$) generalise well.

Key Quantities

- $I(W; S) = H(W) - H(W|S)$: mutual information between weights and training data.
- $I(W; Z_i)$: per-example mutual information, where $Z_i = (\mathbf{x}_i, y_i)$.
- **Individual sample bound:** $\mathbb{E}[R(W) - \hat{R}(W)] \leq \sqrt{\frac{2\sigma^2}{n} \sum_{i=1}^n I(W; Z_i)}$.

SGD as an information bottleneck. The stochastic nature of SGD limits $I(W; S)$: randomness in the mini-batch selection prevents the algorithm from over-memorising specific training examples. Formally, if the SGD noise at step t has variance σ_t^2 , then $I(W; S) \leq \sum_t \frac{\eta_t^2 \|\nabla \mathcal{L}\|^2}{2\sigma_t^2}$, which decreases as noise increases (large σ_t) or step size decreases (small η_t).

Tishby’s information bottleneck. Tishby & Schwartz-Ziv (2017) proposed that neural network training proceeds in two phases: (1) an *error minimisation* phase where $I(T; Y)$ (information about the label) increases, and (2) a *compression* phase where $I(T; \mathbf{X})$ (information about the input) decreases. The compressing representation T lies on the *information bottleneck curve*:

$$\mathcal{IB} : \max_{p(T|X)} I(T; Y) - \beta I(T; X)$$

This picture remains controversial: compression is not universally observed and depends on the activation function and binning method.

EXAMPLE — Bounding Generalisation via Mutual Information

For SGD with Gaussian noise $\square_t \sim \mathcal{N}(0, \sigma^2 \mathbf{I})$ at each step:

$$I(W; S) \leq \frac{\eta^2}{2\sigma^2} \sum_{t=1}^T \mathbb{E} \|\nabla \mathcal{L}(\square_t)\|^2$$

For a well-trained network with gradient norm ~ 0.01 , learning rate $\eta = 0.01$, and noise $\sigma = 0.1$:

$$I(W; S) \leq \frac{(0.01)^2}{2(0.01)} T \times (0.01)^2 = 5 \times 10^{-8} T$$

For $T = 10^5$ steps: $I(W; S) \lesssim 5 \times 10^{-3}$ bits — remarkably small, consistent with good generalisation.

Algorithmic stability. An alternative information-theoretic approach uses *algorithmic stability*: algorithm $\mathcal{A}$ is ϵ -uniformly stable if replacing one training example changes the output by at most ϵ in loss. Stability and mutual information are related: $I(W; Z_i) \leq n\epsilon^2/(2\sigma^2)$ for ϵ -stable algorithms with sub-Gaussian losses. SGD is $O(\eta T/n)$ -stable under standard smoothness assumptions.

The PAC-Bayes bound has a striking interpretation: the generalisation gap is controlled by the *information* the weights carry about the training set. The KL term $\text{KL}(Q\|P)$ measures the “number of bits” needed to describe the posterior relative to the prior. Compressing the model (smaller KL) provably improves generalisation — this is the information-theoretic generalisation viewpoint.

41.4 Algorithmic Stability and Uniform Stability Bounds

Algorithmic stability provides a complementary framework to PAC-Bayes for understanding generalisation: instead of bounding the KL divergence between posterior and prior, it bounds how sensitive the output is to individual training examples.

Uniform Stability

A learning algorithm $\mathcal{A}$ is β -uniformly stable if for any two datasets S, S' that differ in at most one example, and for any test example z :

$$\sup_z |\ell(\mathcal{A}(S), z) - \ell(\mathcal{A}(S'), z)| \leq \beta$$

Stability implies generalisation. Bousquet & Elisseeff (2002) prove: for a β -uniformly stable algorithm with bounded loss $\ell \in [0, 1]$, with probability $\geq 1 - \delta$:

$$R(\mathcal{A}(S)) \leq \hat{R}(\mathcal{A}(S)) + \beta + (4n\beta + 1) \sqrt{\frac{\ln(1/\delta)}{2n}}$$

For $\beta = O(1/n)$: the bound is $O(1/\sqrt{n})$, matching the optimal statistical rate.

Stability of SGD. Hardt, Recht & Singer (2016) establish that SGD is uniformly stable with:

$$\beta \leq \frac{2c^2}{n} \sum_{t=1}^T \eta_t$$

where c is the loss Lipschitz constant and η_t is the step size at step t . For polynomially decaying step sizes $\eta_t = O(1/t)$ and T steps: $\beta = O(\log T/n)$. For constant step sizes η : $\beta = O(\eta T/n)$.

Stability and number of epochs. The stability bound increases with training time T : more SGD steps = less stable = higher generalisation bound. This formalises the empirical observation that early stopping improves generalisation. The optimal stopping time balances the training error reduction (faster convergence = more steps) against the stability penalty (more steps = higher β):

$$T^* \approx \sqrt{n/(c^2\eta)} \implies \beta \approx c\sqrt{\eta/n}, \quad \text{generalisation gap} \approx 2\beta \approx 2c\sqrt{\eta/n}$$

Comparison: stability vs. PAC-Bayes. The two frameworks give complementary bounds:

Framework	Bound	Depends on
Stability	$O(\eta T/n)$	Step size, epochs, n
PAC-Bayes	$O(\sqrt{\text{KL}(Q\ P)/n})$	Compressibility, n
VC theory	$O(\sqrt{d_{\text{VC}}/n})$	Hypothesis class only
Rademacher	$O(\mathfrak{R}_n(\mathcal{F}))$	Data distribution + class

For overparameterised networks: VC bounds are vacuous; stability and PAC-Bayes give the tightest known non-vacuous bounds. They are complementary: stability is tight for small T , PAC-Bayes is tight when the posterior is highly compressible.

EXAMPLE — Stability Bound for ResNet-50 on ImageNet

$n = 1.28 \times 10^6$, $T = 90 \times 1280000/256 \approx 4.5 \times 10^5$ steps, $\eta = 0.1/(1 + t/10^5)$ (warmup + decay).

$\sum_t \eta_t \approx 10^5 \ln(4.5) + \int_{10^5}^{4.5 \times 10^5} \frac{0.1 \cdot 10^5}{t} dt \approx 10^5(1.5 + 0.1 \ln 4.5) \approx 1.65 \times 10^5$.

Lipschitz constant $c \approx 1$ (cross-entropy with softmax has $c \leq 1$).

Stability: $\beta \leq 2 \times 10^2 \times 1.65 \times 10^5 / (1.28 \times 10^6) \approx 0.26$.

Generalisation bound: $R \leq \hat{R} + 0.26 + O(1/\sqrt{n}) \approx \hat{R} + 0.26$.

This is non-vacuous but loose: actual generalisation gap is ~ 0.01 . The stability bound overestimates because it assumes worst-case sensitivity, while in practice most steps change the output by much less than β .

41.5 Double Descent and the Interpolation Threshold

Classical statistical learning theory predicts a U-shaped bias-variance tradeoff: models with too few parameters underfit (high bias), models with too many overfit (high variance). Modern deep learning violates this prediction, exhibiting a *double descent* curve.

The double descent phenomenon. Belkin et al. (2019) and Nakkiran et al. (2020) observe that as model size N increases beyond the interpolation threshold ($N = n$, where the model can exactly fit the training data), the test error *decreases* again after an initial increase at the threshold:

- $N < n$ (classical regime): test error follows the U-curve; bias dominates at small N , variance at large N .
- $N = n$ (interpolation threshold): the model exactly interpolates training data; variance is maximal (many solutions exist; gradient descent picks one unpredictably).
- $N > n$ (overparameterised regime): test error decreases as N increases; the minimum-norm interpolator generalises better as the model becomes more over-parameterised.

Kernel regression perspective. For the kernel regression estimator $\hat{f}(\mathbf{x}) = \mathbf{k}(\mathbf{x}, \mathbf{X})\mathbf{K}^{-1}\mathbf{Y}$ with kernel matrix $\mathbf{K}_{ij} = k(\mathbf{x}_i, \mathbf{x}_j)$:

- As the kernel bandwidth $\sigma \rightarrow 0$: the kernel becomes a delta function; the estimator interpolates but has high variance. $N \rightarrow \infty$ limit: minimum-norm kernel interpolator.
- As the kernel bandwidth $\sigma \rightarrow \infty$: the kernel becomes flat; the estimator is the constant predictor. High bias, zero variance.

The double descent in neural networks corresponds to the transition from “finite width” (kernel regime fails, many bad interpolators) to “infinite width” (NTK kernel, consistent

interpolation).

Benign overfitting conditions. Bartlett et al. (2020) characterise when interpolating estimators generalise well. For ridge regression $\hat{\theta} = (\mathbf{X}^\top \mathbf{X} + \lambda \mathbf{I})^{-1} \mathbf{X}^\top \mathbf{y}$ with $\lambda \rightarrow 0$:

Benign Overfitting (Bartlett et al., 2020)

For a linear model with data covariance Σ (eigenvalues $\lambda_1 \geq \lambda_2 \geq \dots$), the minimum-norm interpolator $\hat{\theta}$ satisfies:

$$\mathbb{E} \|\hat{\theta} - \theta^*\|_\Sigma^2 = O\left(\frac{k_*(\Sigma)}{n}\right)$$

where $k_*(\Sigma) = \frac{(\sum_{i>k} \lambda_i)^2}{\sum_{i>k} \lambda_i^2}$ is the *effective rank* of the tail of Σ , for any k such that $\sum_{i \leq k} \lambda_i = O(n\lambda_{k+1})$.

Corollary: Benign overfitting occurs when $k_*(\Sigma) = \Omega(n)$ (the tail has effective rank at least n). This is a “spiked covariance” condition: a few large eigenvalues carry the signal; many small eigenvalues carry noise that the minimum-norm solution distributes across.

Epoch-wise double descent. Nakkiran et al. (2020) observe a temporal version: for a fixed model size, test error first decreases, then increases (overfitting), then decreases again with more training epochs. This epoch-wise double descent is explained by the stability theory (§41.4): at the interpolation epoch, the model is maximally unstable (β is large), and further training finds a more stable (lower-norm) solution.

EXAMPLE — Double Descent on CIFAR-10 (ResNet-18)

Varying number of features (channels multiplied by width factor w):

Width w	Parameters	Test Error
0.1	11 k	24.5%
0.3	99 k	17.2%
0.5	275 k (threshold)	21.8% (peak!)
1.0	1.1 M	14.5%
3.0	9.9 M	10.2%
10.0	110 M	9.8%

At $w = 0.5$ (near interpolation threshold $N \approx n = 50000$): test error peaks at 21.8%. Both smaller and larger models achieve lower error. The peak is the “interpolation threshold catastrophe” predicted by double descent theory.

Information-Theoretic Learning: Mutual Information and IB

The Information Bottleneck principle frames representation learning as a rate-distortion problem: compress the input while preserving the label.

42.1 Mutual Information

Mutual information (MI) is the cornerstone quantity of information theory, measuring the statistical dependence between two random variables. In the context of neural networks, it appears in generalisation bounds, representation learning objectives, and the analysis of what networks learn about their inputs.

Mutual Information

For discrete random variables X and Y with joint distribution $p(x, y)$:

$$I(X; Y) = \sum_{x, y} p(x, y) \log \frac{p(x, y)}{p(x)p(y)} = H(X) - H(X|Y) = H(Y) - H(Y|X)$$

For continuous variables, replace sums with integrals (differential entropy). Key properties:

- $I(X; Y) \geq 0$, with equality iff $X \perp Y$.
- $I(X; Y) = \text{KL}(p(x, y) \| p(x)p(y))$.
- $I(X; Y) = I(Y; X)$ (symmetry).
- **Data processing inequality:** if $X \rightarrow Y \rightarrow Z$ is a Markov chain, $I(X; Z) \leq I(X; Y)$.

Estimating mutual information. For high-dimensional neural representations, exact MI computation is intractable. Several estimators are used in practice:

MINE (Mutual Information Neural Estimation) (Belghazi et al. 2018) uses the Donsker-Varadhan representation:

$$I(X; Y) = \sup_{T: \mathcal{X} \times \mathcal{Y} \rightarrow \mathbb{R}} \mathbb{E}_{p(x, y)}[T(x, y)] - \log \mathbb{E}_{p(x)p(y)}[e^{T(x, y)}]$$

The supremum is achieved by $T^*(x, y) = \log \frac{p(x, y)}{p(x)p(y)} + c$. MINE parametrises T as a neural network and optimises via SGD.

InfoNCE uses contrastive estimation:

$$I(X; Y) \geq \mathbb{E} \left[\log \frac{e^{f(x, y)}}{\frac{1}{K} \sum_{k=1}^K e^{f(x, y_k)}} \right] + \log K$$

where $y_1 = y$ is the positive sample and $y_2, \dots, y_K$ are negatives. This bound is tight as $K \rightarrow \infty$ and is the loss used in contrastive representation learning (SimCLR, MoCo).

EXAMPLE — Mutual Information in SimCLR

SimCLR maximises $I(z_1; z_2)$ where z_1, z_2 are two augmented views of the same image.

With batch size $K = 4096$ and InfoNCE:

$$\mathcal{L}_{\text{SimCLR}} = -\frac{1}{2K} \sum_{k=1}^K \left[\log \frac{e^{z_k^\top z'_k / \tau}}{\sum_{j \neq k} e^{z_k^\top z'_j / \tau}} + \log \frac{e^{z'^{\top}_k z_k / \tau}}{\sum_{j \neq k} e^{z'^{\top}_k z'_j / \tau}} \right]$$

The temperature τ controls the tightness of the InfoNCE bound. At convergence, $I(z_1; z_2) \geq \log 4096 \approx 12$ bits.

MI and the neural tangent kernel. The mutual information $I(f_\theta(\mathbf{X}); \mathbf{Y})$ quantifies the predictive power of a network f_θ . For Gaussian processes (the infinite-width NTK limit), this becomes:

$$I(f_\theta(\mathbf{X}); \mathbf{Y}) = \frac{1}{2} \log \det(\mathbf{I} + \sigma^{-2} \mathbf{K})$$

where $\mathbf{K}$ is the NTK Gram matrix. Maximising this quantity with respect to the architecture θ leads to *maximum mutual information network design*.

Mutual Information

For random variables X and Y :

$$I(X; Y) = \sum_{x,y} p(x, y) \log \frac{p(x, y)}{p(x)p(y)} = H(Y) - H(Y|X) = H(X) - H(X|Y)$$

$I(X; Y) \geq 0$, with equality iff $X \perp Y$. It measures how much knowing X reduces uncertainty about Y .

For continuous variables, replace sums with integrals and log with the differential entropy.

42.2 The Information Bottleneck

The Information Bottleneck (IB) method (Tishby, Pereira & Bialek, 1999) is a framework for learning minimal sufficient representations: finding a compressed variable Z that retains as much information about the target Y as possible while discarding irrelevant information from the input X .

The IB objective. Given a joint distribution $p(X, Y)$, find an encoding $p(Z|X)$ that solves:

$$\min_{p(Z|X)} I(X; Z) - \beta I(Z; Y)$$

where $I(X; Z)$ is the mutual information between input and representation (compression), $I(Z; Y)$ is the mutual information between representation and target (relevance), and $\beta > 0$ is a Lagrange multiplier trading off compression against relevance.

Formal solution via Lagrangian. The IB Lagrangian relaxes the constraint $I(Z; Y) \geq R_Y$:

$$\mathcal{F}[p(Z|X)] = I(X; Z) - \beta I(Z; Y) = H(Z) - H(Z|X) - \beta[H(Y) - H(Y|Z)]$$

Since $H(Y)$ is fixed, this reduces to minimising $H(Z) - H(Z|X) + \beta H(Y|Z)$. The optimal encoder satisfies the self-consistent equations (Blahut-Arimoto-style iteration):

$$\begin{aligned} p^*(z|x) &\propto p(z) \exp(-\beta D_{\text{KL}}(p(y|x) \| p(y|z))) \\ p^*(z) &= \int p^*(z|x) p(x) dx \\ p^*(y|z) &= \frac{\int p(y|x) p^*(z|x) p(x) dx}{p^*(z)} \end{aligned}$$

The IB curve. As β varies from 0 to ∞ :

- $\beta = 0$: no relevance term; Z is maximally compressed ($Z = \text{constant}$, $I(X; Z) = 0$).
- $\beta \rightarrow \infty$: no compression; $Z = X$, $I(X; Z) = H(X)$, $I(Z; Y) = I(X; Y)$.
- Intermediate β : traces out the
emphinformation curve $I(Z; Y)$ vs. $I(X; Z)$.

The information curve is concave (by data processing inequality) and lies below the line $I(Z; Y) = I(X; Y)$ (perfect relevance at full information).

EXAMPLE — Information Bottleneck for a Gaussian Mixture

Let $X \sim \mathcal{N}(0, \Sigma_X)$, $Y = \mathbf{A}X + \epsilon$ with $\epsilon \perp X$. The optimal IB encoder is a linear projection:

$$Z = \mathbf{B}X, \quad \mathbf{B} = \text{top-}k \text{ eigenvectors of } \Sigma_X^{1/2} \mathbf{A}^\top (\mathbf{A} \Sigma_X \mathbf{A}^\top + \Sigma_\epsilon)^{-1} \mathbf{A} \Sigma_X^{1/2}$$

The k retained dimensions correspond to the k most informative directions about Y . For β large enough to retain k directions: $I(X; Z) = -\frac{1}{2} \sum_{i=1}^k \ln(1 - \rho_i^2)$ where ρ_i are canonical correlations.

The information bottleneck (IB) framework, introduced by Tishby, Pereira & Bialek (1999), formalises the trade-off between compression and prediction. It provides a principled way to extract the minimal sufficient statistics of $\mathbf{X}$ for predicting Y .

The IB objective. Given a joint distribution $p(\mathbf{x}, y)$, find a compressed representation T of $\mathbf{X}$ that is maximally informative about Y :

$$\min_{p(t|\mathbf{x})} I(T; \mathbf{X}) - \beta I(T; Y)$$

The Lagrange multiplier $\beta > 0$ controls the compression-relevance trade-off. Large β prioritises predictive power; small β prioritises compression.

Information Bottleneck Curve

The IB curve is the Pareto frontier of achievable $(I(T; \mathbf{X}), I(T; Y))$ pairs:

$$I^*(R_X) = \sup_{p(t|\mathbf{x}): I(T; \mathbf{X}) \leq R_X} I(T; Y)$$

Points on this curve are called *sufficient* representations: they retain all predictive information about Y at each compression level R_X .

Self-consistent equations. The optimal IB solution satisfies the Blahut-Arimoto-like equations:

$$p^*(t|\mathbf{x}) = \frac{p(t)}{Z(\mathbf{x}, \beta)} e^{-\beta \text{KL}(p(y|\mathbf{x}) \| p(y|t))}$$

$$p^*(y|t) = \frac{1}{p(t)} \sum_{\mathbf{x}} p(y|\mathbf{x}) p(t|\mathbf{x}) p(\mathbf{x})$$

These equations alternate between updating the encoder $p(t|\mathbf{x})$ and the decoder $p(y|t)$, converging to a local optimum of the IB Lagrangian.

Variational IB for deep learning. Since exact IB is intractable for continuous, high-dimensional $\mathbf{X}$, Alemi et al. (2017) proposed the *variational information bottleneck* (VIB):

$$\mathcal{L}_{\text{VIB}} = \mathbb{E}_{\mathbf{x}, y} \left[\mathbb{E}_{\mathbf{z} \sim q(\mathbf{z}|\mathbf{x})} [-\log p(y|\mathbf{z})] + \beta \text{KL}(q(\mathbf{z}|\mathbf{x}) \| p(\mathbf{z})) \right]$$

With $q(\mathbf{z}|\mathbf{x}) = \mathcal{N}(\mu_\theta(\mathbf{x}), \sigma_\theta^2(\mathbf{x})\mathbf{I})$ and $p(\mathbf{z}) = \mathcal{N}(0, \mathbf{I})$, the KL term has a closed form:

$$\text{KL}(q \| p) = \frac{1}{2} \sum_j [\mu_j^2 + \sigma_j^2 - 1 - \log \sigma_j^2]$$

The VIB thus becomes a β -VAE with a classification head — a natural bridge between representation learning and information theory.

EXAMPLE — VIB on MNIST

With $\beta = 0.01$ and a 256-dimensional latent space:

- $I(Z; \mathbf{X}) \approx 14$ bits (moderate compression of 784-dim input).
- $I(Z; Y) \approx 3.32$ bits = $\log_2 10$ (perfect prediction of 10 classes).
- Test accuracy: 99.1%.

Increasing $\beta \rightarrow 0.1$: $I(Z; \mathbf{X})$ drops to 5 bits, test accuracy drops to 97.8%, but the model is significantly more robust to input perturbations.

IB and adversarial robustness. Representations that minimise $I(T; \mathbf{X})$ at fixed $I(T; Y)$ are, by construction, insensitive to input variations that are irrelevant to Y . This explains the empirical observation that VIB-trained models exhibit improved adversarial robustness: the encoder cannot transmit adversarial perturbations to the classifier if they carry no information about Y .

Given the Markov chain $Y \rightarrow X \rightarrow Z$ (label, input, representation), the IB objective balances compression and relevance:

$$\min_{p(z|x)} I(X; Z) - \beta I(Z; Y)$$

$\beta > 0$ trades compression (small $I(X; Z)$) against relevance (large $I(Z; Y)$).

IB self-consistent equations. The optimal encoder $p^*(z|x)$ satisfies:

$$\begin{aligned} p^*(z|x) &= \frac{p(z)}{Z(x, \beta)} \exp(-\beta D_{\text{KL}}[p(y|x) \| p(y|z)]) \\ p^*(z) &= \int p^*(z|x) p(x) dx \\ p^*(y|z) &= \frac{p(y) p^*(z|y)}{p^*(z)} \end{aligned}$$

These are solved by alternating Blahut–Arimoto iterations.

42.3 Tishby’s Neural Network Interpretation

Tishby & Schwartz-Ziv (2017) proposed that neural network training has two distinct phases, visible when plotting the mutual information $I(X; T^{(l)})$ and $I(T^{(l)}; Y)$ for each layer l during training:

Phase 1 — Empirical risk minimisation (ERM). In early training, both $I(X; T)$ and $I(T; Y)$ increase for all layers. The network is learning to encode input information that is predictive of the output.

Phase 2 — Compression. After sufficient ERM, $I(X; T)$ begins to *decrease* for hidden layers while $I(T; Y)$ remains stable. The network is “forgetting” irrelevant information about X while retaining what matters for Y .

The compression-generalisation connection. The IB bound on test error is:

$$R_{\text{test}} \leq \hat{R} + O\left(\sqrt{\frac{I(X; T)}{n}}\right)$$

A smaller $I(X; T)$ means better generalisation. Compression thus provides a mechanistic explanation for why networks generalise: SGD noise during training acts as a diffusion process that naturally reduces $I(X; T)$ in stochastic layers.

Formal analysis of the compression phase. Under the Gaussian data model (inputs $\mathbf{X} \sim \mathcal{N}(0, \mathbf{I})$, labels $Y = \mathbf{w}^* \top \mathbf{X} + \varepsilon$), the mutual information between a linear representation $\mathbf{Z} = \mathbf{B}\mathbf{X}$ and the input is:

$$I(\mathbf{X}; \mathbf{Z}) = -\frac{1}{2} \log \det(\mathbf{I} - \mathbf{B}(\mathbf{B}^\top \mathbf{B})^{-1} \mathbf{B}^\top) = -\frac{1}{2} \log \det(\mathbf{I} - \mathbf{P}_\mathbf{B})$$

where $\mathbf{P}_\mathbf{B}$ is the projection onto the row space of $\mathbf{B}$. During gradient descent, $I(\mathbf{X}; \mathbf{Z})$ decreases only if the projection matrix $\mathbf{P}_\mathbf{B}$ changes (some directions of $\mathbf{X}$ become uncaptured). For linear networks with squared loss, the gradient flow preserves the row space of $\mathbf{B}$ once it has converged — hence no compression for linear activations.

Stochasticity as the driver of compression. For non-linear networks with mini-batch SGD, the stochastic gradient introduces noise $\square_t \sim \mathcal{N}(0, \Sigma_t)$ at each step. This noise diffuses through the hidden layers, effectively adding a stochastic channel $\mathbf{Z} = \mathbf{B}\mathbf{X} + \square$. The mutual information becomes:

$$I(\mathbf{X}; \mathbf{Z}) = H(\mathbf{Z}) - H(\mathbf{Z}|\mathbf{X}) = H(\mathbf{B}\mathbf{X} + \square) - H(\square)$$

As the noise $\square$ increases (from larger gradient noise or smaller batch size), $I(\mathbf{X}; \mathbf{Z})$ decreases — this is the compression phase. The compression phase is therefore a *regularisation effect* of stochastic training, not an intrinsic property of neural network dynamics.

Implications for architecture design. The IB perspective suggests:

- Bottleneck layers (physically small hidden dimensions) enforce $I(\mathbf{X}; \mathbf{Z}) \leq \log |\mathbf{Z}|$, encouraging the network to learn minimal sufficient representations.
- Variational autoencoders (§17 in Volume II) are explicit implementations of the IB: the KL term bounds $I(\mathbf{X}; \mathbf{Z})$ and the reconstruction term maximises $I(\mathbf{Z}; \mathbf{X})$.
- Contrastive learning objectives (SimCLR, MoCo) maximise $I(\mathbf{Z}_1; \mathbf{Z}_2)$ between two augmented views, equivalent to finding representations that retain class-relevant information while discarding view-specific noise.

The controversy. Saxe et al. (2019) showed that the compression phase depends critically on the activation function and the mutual information estimator used. For ReLU activations with deterministic training (full-batch gradient descent), no compression phase is observed. The compression phase arises specifically for non-linear activations in the presence of stochasticity (mini-batch noise), which acts as the implicit diffusion. The debate about whether IB explains deep learning generalisation remains active.

EXAMPLE — Information Plane Dynamics, 7-Layer Tanh Network

Training on a synthetic dataset ($n = 5000$, $d = 12$ inputs, binary label), mutual information estimated via binning the hidden layer activations.

Epoch	$I(X; T_3)$	$I(T_3; Y)$	Phase
0	2.1	0.1	Init
100	4.8	0.7	ERM
500	5.2	0.92	ERM
2000	3.1	0.92	Compression
5000	1.4	0.91	Compression

$I(X; T_3)$ drops from 5.2 to 1.4 bits while $I(T_3; Y)$ stays at 0.91 bits. The layer compresses irrelevant input information without losing label-relevant information — consistent with the IB objective.

Tishby and Schwartz-Ziv (2017) applied IB to the intermediate layers of a DNN. Each layer Z_l is a compressed representation of X ; the “information plane” plots $I(X; Z_l)$ vs $I(Z_l; Y)$. During training:

1. **Fitting phase.** $I(X; Z_l)$ and $I(Z_l; Y)$ both increase: the layer memorises the training data.
2. **Compression phase.** $I(Z_l; Y)$ stays roughly constant while $I(X; Z_l)$ decreases: the layer forgets irrelevant details.

This two-phase story is debated. The compression phase is sensitive to the mutual information estimator used; with deterministic activations (ReLU), $I(X; Z)$ may not decrease. Nevertheless, the IB framework provides a principled language for representation learning.

42.4 Mutual Information Neural Estimation (MINE)

Computing mutual information $I(X; Z)$ is intractable for high-dimensional X and Z . MINE (Belghazi et al., 2018) uses a neural network to estimate mutual information via the Donsker-Varadhan variational formula.

Donsker-Varadhan representation. For any joint distribution P_{XZ} with marginals P_X, P_Z :

$$I(X; Z) = \sup_{T: \mathcal{X} \times \mathcal{Z} \rightarrow \mathbb{R}} \mathbb{E}_{P_{XZ}}[T] - \ln \mathbb{E}_{P_X \otimes P_Z}[e^T]$$

The supremum is achieved by $T^*(x, z) = \ln \frac{dP_{XZ}}{d(P_X \otimes P_Z)}(x, z) = \ln \frac{p(x, z)}{p(x)p(z)}$ (the log density ratio).

MINE estimator. Parameterise $T_\theta : \mathcal{X} \times \mathcal{Z} \rightarrow \mathbb{R}$ as a neural network. Given samples from P_{XZ} and the marginal product $P_X \otimes P_Z$:

$$\hat{I}(X; Z) = \sup_{\theta} \frac{1}{n} \sum_{i=1}^n T_\theta(x_i, z_i) - \ln \left(\frac{1}{n} \sum_{j=1}^n e^{T_\theta(x_j, \bar{z}_j)} \right)$$

where $\bar{z}_j$ are samples from the marginal P_Z (obtained by shuffling the z_i).

Bias-corrected MINE. The direct estimator is biased due to the log of expectation. The exponential moving average (EMA) estimator reduces bias:

$$\hat{I}_{\text{EMA}} = \mathbb{E}[T_\theta] - \frac{\mathbb{E}[e^{T_\theta} \cdot T_\theta]_{\text{EMA}}}{\mathbb{E}[e^{T_\theta}]_{\text{EMA}}}$$

EXAMPLE — MINE for the Information Plane

For a 4-layer ReLU network on a synthetic task, computing $I(X; Z^{(l)})$ for each layer l using MINE:

- Input $X \in \mathbb{R}^{10}$, hidden layers $Z^{(l)} \in \mathbb{R}^{20}$, label $Y \in \{0, 1\}^5$.
- MINE neural network: 3-layer MLP with 256 hidden units.
- Training: 5000 gradient steps per layer per epoch.
- Estimated $I(X; Z^{(1)})$ converges to within 3% of the true value (verified on a Gaussian problem where MI is known analytically).

MINE enables the information plane analysis of §42.3 in practice, despite X and Z being high-dimensional continuous random variables.

42.5 Variational Inference Meets Information Theory

The variational information bottleneck connects the IB framework to variational autoencoders and provides practical algorithms for learning minimal sufficient representations.

Dual encoder architecture. The VIB framework uses two encoders:

- *Encoder:* $q_\phi(\mathbf{z}|\mathbf{x}) = \mathcal{N}(\mu_\phi(\mathbf{x}), \Sigma_\phi(\mathbf{x}))$ maps input $\mathbf{x}$ to a stochastic latent code $\mathbf{z}$.
- *Classifier:* $p_\theta(y|\mathbf{z}) = \text{softmax}(f_\theta(\mathbf{z}))$ predicts label from latent code.
- *Prior:* $p(\mathbf{z}) = \mathcal{N}(\mathbf{0}, \mathbf{I})$ (marginal over inputs).

The VIB loss:

$$\mathcal{L}_{\text{VIB}}(\phi, \theta) = \mathbb{E}_{\mathbf{x}, y} \left[\mathbb{E}_{\mathbf{z} \sim q_\phi(\cdot|\mathbf{x})} [-\log p_\theta(y|\mathbf{z})] + \beta \text{KL}(q_\phi(\mathbf{z}|\mathbf{x}) \| p(\mathbf{z})) \right]$$

minimises cross-entropy (task performance) while penalising the KL divergence (compression). Increasing β forces stronger compression.

Information plane trajectory. During training with VIB, the encoder traces a path on the information plane $\{(I(\mathbf{X}; \mathbf{Z}), I(\mathbf{Z}; Y))\}$:

1. Early training (high β): strong compression forces $\mathbf{Z} \approx \mathbf{0}$ (constant); $I(\mathbf{X}; \mathbf{Z}) \approx 0$, $I(\mathbf{Z}; Y) \approx 0$.
2. Intermediate training: encoder learns the most predictive directions of $\mathbf{X}$ for Y ; both MI increase.
3. Convergence: $(I(\mathbf{X}; \mathbf{Z}), I(\mathbf{Z}; Y))$ approaches the information curve from below.

The value of β determines where on the IB curve the final representation lies.

KL decomposition. The KL term decomposes as:

$$\text{KL}(q_\phi(\mathbf{z}|\mathbf{x}) \| p(\mathbf{z})) = I_q(\mathbf{X}; \mathbf{Z}) + \text{KL}(q_\phi(\mathbf{z}) \| p(\mathbf{z}))$$

where I_q is the MI under the encoder q and $q_\phi(\mathbf{z}) = \int q_\phi(\mathbf{z}|\mathbf{x})p(\mathbf{x})d\mathbf{x}$ is the aggregate posterior. The second term $\text{KL}(q_\phi(\mathbf{z})\|p(\mathbf{z}))$ vanishes when the aggregate posterior matches the prior — the aggregation regularisation. Minimising the full KL thus minimises $I_q(\mathbf{X}; \mathbf{Z})$ (compression) while matching the aggregate posterior to the prior (regularisation).

Practical advantages of VIB over standard cross-entropy.

- *Calibration*: the stochastic encoder produces uncertainty estimates; $\text{Var}(\mathbf{z})$ is large for out-of-distribution inputs.
- *Adversarial robustness*: adding Gaussian noise to the representation (KL term) prevents the encoder from transmitting adversarial perturbations.
- *Feature disentanglement*: high β forces each dimension of $\mathbf{z}$ to encode a single independent factor (similar to β -VAE disentanglement).

EXAMPLE — VIB vs. Standard Training: Adversarial Robustness

MNIST, $\beta = 0.01$ VIB vs. standard cross-entropy:

Method	Clean Accuracy	FGSM Accuracy ($\epsilon = 0.1$)
Standard (CE)	99.4%	56.2%
VIB ($\beta = 10^{-3}$)	99.3%	73.8%
VIB ($\beta = 10^{-2}$)	99.1%	82.1%
VIB ($\beta = 10^{-1}$)	97.8%	88.4%
Adversarial training (PGD)	99.0%	95.1%

VIB achieves substantially higher FGSM accuracy than standard training without explicit adversarial training, by discarding the high-frequency adversarial features through compression. Full adversarial training remains superior, but VIB provides a computationally cheap robustness improvement.

Nonlinear IB and optimal transport. For non-Gaussian distributions, the optimal IB encoder can be computed via:

1. Estimate $p(\mathbf{x}, y)$ from data.
2. Solve the IB Lagrangian using the Blahut-Arimoto algorithm (alternating optimisation of encoder and decoder).
3. The optimal encoder is a deterministic function of the ratio $p(y|\mathbf{x})/p(y|\mathbf{z})$ — a sufficient statistic.

For Gaussian models, the optimal encoder projects onto the top canonical directions (maximising $\sum \rho_i^2/(1 - \rho_i^2)$). For nonlinear models, neural networks parametrise the encoder q_ϕ and the Blahut-Arimoto iterations become stochastic gradient descent steps.

Adversarial Attacks: FGSM and PGD

Small, imperceptible perturbations can catastrophically degrade neural network predictions. We derive the Fast Gradient Sign Method and the Projected Gradient Descent attack from first principles.

43.1 The Adversarial Example Problem

Adversarial examples are inputs $\mathbf{x}' = \mathbf{x} + \square$ that are imperceptibly different from a legitimate input $\mathbf{x}$ but cause a neural network to produce a wildly incorrect output. Discovered by Szegedy et al. (2014), they reveal a fundamental tension between the human-perceived geometry of input space and the learned decision boundaries of deep networks.

Formal definition. Given a classifier $f_\theta : \mathcal{X} \rightarrow \mathcal{Y}$ and an input $\mathbf{x}$ with true label y , an adversarial example is $\mathbf{x}' = \mathbf{x} + \square$ such that:

$$f_\theta(\mathbf{x}') \neq y \quad \text{and} \quad \|\square\|_p \leq \epsilon$$

for some small $\epsilon > 0$ and ℓ_p norm. Common choices:

- ℓ_∞ : each pixel changes by at most ϵ (often $\epsilon = 8/255$).
- ℓ_2 : total perturbation energy bounded.
- ℓ_0 : at most k pixels may change (sparse attacks).

Targeted vs. Untargeted Attacks

- **Untargeted:** maximise $\mathcal{L}(f_\theta(\mathbf{x} + \square), y)$ — cause any misclassification.
- **Targeted:** minimise $\mathcal{L}(f_\theta(\mathbf{x} + \square), y_{\text{target}})$ — cause a specific misclassification.

Targeted attacks are strictly harder: they require finding a perturbation that steers the output to a specific class rather than simply away from the correct one.

Why adversarial examples exist. Goodfellow et al. (2015) provided an early explanation via the *linearity hypothesis*: in high dimensions, a linear model $f(\mathbf{x}) = \mathbf{w}^\top \mathbf{x}$ can change significantly from a small perturbation because the perturbation $\square$ aligns with $\mathbf{w}$ across all d dimensions:

$$f(\mathbf{x} + \square) - f(\mathbf{x}) = \mathbf{w}^\top \square \leq \|\mathbf{w}\|_1 \cdot \|\square\|_\infty$$

For $d = 10^4$ pixels with $\epsilon = 8/255$ and $\mathbf{w} = \mathbf{1}/\sqrt{d}$: change $\approx \sqrt{d}\epsilon \approx 3$, large enough to cross a decision boundary.

Non-robustness vs. misaligned features. Ilyas et al. (2019) argued that adversarial examples exploit *non-robust features*: predictive but brittle patterns (e.g., texture correlations) that are genuinely informative for in-distribution classification but fail under distribution shift. Under this view, adversarial vulnerability is not a bug but a feature — the network has learned all available signal, including fragile correlations.

EXAMPLE — Adversarial Vulnerability of ImageNet Models

On a standard ResNet-50 trained on ImageNet:

- Clean accuracy: 76.1%.
- After FGSM ($\epsilon = 8/255, \ell_\infty$): 5.2%.

- After PGD-20 ($\epsilon = 8/255$): 0.0%.
- After adversarial training (PGD-7, $\epsilon = 8/255$): PGD-20 accuracy = 49.0%.

The dramatic drop from clean to adversarial accuracy — from 76% to 0% — illustrates the severity of the problem. Adversarial training recovers roughly half the gap but at a 10–15% clean accuracy cost.

Carlini-Wagner (C&W) attack. Carlini & Wagner (2017) formulate adversarial examples as a constrained optimisation problem with a carefully designed objective that avoids gradient masking artifacts:

$$\min_{\square} \|\square\|_2^2 + c \cdot f(\mathbf{x} + \square, y_{\text{target}})$$

where $f(\mathbf{x}', y) = \max(\max_{j \neq y} \log p_j(\mathbf{x}') - \log p_y(\mathbf{x}'), -\kappa)$ encourages misclassification with margin κ . The change of variable $\square = \frac{1}{2}(\tanh(\mathbf{w}) + 1) - \mathbf{x}$ automatically enforces $[0, 1]$ pixel constraints. C&W is typically stronger than PGD and is used to detect certified defences (models that appear robust to PGD but have gradient masking may still be broken by C&W).

Gradient masking and obfuscation. Some defences cause gradients to be near-zero or uninformative (“gradient masking”), making FGSM/PGD appear to fail while the model remains vulnerable to other attacks. Common masking mechanisms:

- *Shattered gradients*: non-differentiable pre-processing (quantisation, JPEG compression) makes gradients zero almost everywhere.
- *Stochastic gradients*: random perturbations at test time randomise the gradient direction.
- *Exploding/vanishing gradients*: very deep defences accumulate gradient issues.

AutoAttack (§43.3 extension) addresses gradient masking by combining gradient-based and gradient-free attacks.

AutoAttack. Croce & Hein (2020) propose a standardised, parameter-free evaluation benchmark combining four attacks:

1. APGD-CE: auto step-size PGD with cross-entropy loss.
2. APGD-DLR: APGD with difference of logits ratio loss (invariant to logit scaling).
3. FAB: Fast Adaptive Boundary attack, finds minimum-norm adversarial examples.
4. Square attack: score-based black-box attack (no gradient access required).

AutoAttack is the standard benchmark for adversarial robustness evaluation; a model is considered “robustly evaluated” only if its accuracy is reported under AutoAttack.

EXAMPLE — AutoAttack vs. PGD-20 on RobustBench

Standard model (no adversarial training), ResNet-18, CIFAR-10:

Attack	Clean Acc	Robust Acc ($\epsilon = 8/255$)
None (clean)	93.5%	—
FGSM	93.5%	22.1%
PGD-20	93.5%	0.3%
AutoAttack	93.5%	0.0%

PGD-20 already achieves near-zero robustness for unevaluated models. AutoAttack's additional attacks (FAB, Square) catch the remaining 0.3% that PGD-20 missed.

Adversarially trained model (PGD-7 training, $\epsilon = 8/255$):

Attack	Clean Acc	Robust Acc
None	84.1%	—
PGD-20	84.1%	49.8%
AutoAttack	84.1%	48.2%

The small gap (49.8% vs. 48.2%) shows that adversarial training genuinely increases robustness — not just gradient masking.

Transferability. A striking property of adversarial examples is that perturbations crafted for one model often fool other models with different architectures trained on different data. This transferability enables *black-box attacks*: an adversary without access to the target model can craft perturbations against a surrogate model and transfer them to the target. Transfer rates of 40–80% are common across architectures.

Given a classifier $f_{\square} : \mathcal{X} \rightarrow \mathcal{Y}$ with true label y for input $\mathbf{x}$, an adversarial example is $\mathbf{x}' = \mathbf{x} + \square$ such that:

1. $\|\square\|_{\infty} \leq \epsilon$ (imperceptibly small perturbation), and
2. $f_{\square}(\mathbf{x}') \neq y$ (wrong prediction).

The adversary solves:

$$\max_{\|\square\|_{\infty} \leq \epsilon} \mathcal{L}(f_{\square}(\mathbf{x} + \square), y)$$

43.2 Fast Gradient Sign Method (FGSM)

Goodfellow et al. (2014) proposed a one-step attack based on linearising $\mathcal{L}$ around $\mathbf{x}$:

$$\mathcal{L}(\mathbf{x} + \square) \approx \mathcal{L}(\mathbf{x}) + \square^{\top} \nabla_{\mathbf{x}} \mathcal{L}(\mathbf{x})$$

Under the ℓ_{∞} constraint, the maximiser is:

$$\begin{aligned} \square^* &= \epsilon \cdot \text{sign}(\nabla_{\mathbf{x}} \mathcal{L}(\mathbf{x})) \\ \mathbf{x}' &= \mathbf{x} + \epsilon \cdot \text{sign}(\nabla_{\mathbf{x}} \mathcal{L}(\mathbf{x})) \end{aligned}$$

Derivation. The first-order bound on loss increase is:

$$\square^{\top} \nabla_{\mathbf{x}} \mathcal{L} \leq \|\square\|_{\infty} \cdot \|\nabla_{\mathbf{x}} \mathcal{L}\|_1$$

Equality holds when $\delta_i = \epsilon \cdot \text{sign}([\nabla_{\mathbf{x}} \mathcal{L}]_i)$ for each coordinate i .

EXAMPLE — FGSM on a 3-pixel image

$$\mathbf{x} = (0.5, 0.3, 0.8)^\top, \epsilon = 0.1, \nabla_{\mathbf{x}} \mathcal{L} = (1.2, -0.5, 0.3)^\top.$$

$$\text{sign}(\nabla_{\mathbf{x}} \mathcal{L}) = (1, -1, 1)^\top$$

$$\mathbf{x}' = (0.5 + 0.1, 0.3 - 0.1, 0.8 + 0.1)^\top = (0.6, 0.2, 0.9)^\top$$

Each pixel moves by exactly ϵ in the direction that increases loss. $\|\square\|_\infty = 0.1 = \epsilon$. ✓

43.3 Projected Gradient Descent (PGD)

FGSM is a single step. Madry et al. (2017) proposed iterating gradient steps with projection back onto the ℓ_∞ ball:

PGD Attack (k steps)

Initialise $\mathbf{x}^{(0)} = \mathbf{x} + \text{Uniform}(-\epsilon, \epsilon)^d$. For $t = 0, \dots, k - 1$:

$$\mathbf{x}^{(t+1)} = \Pi_{\mathcal{B}_\infty(\mathbf{x}, \epsilon)}(\mathbf{x}^{(t)} + \alpha \cdot \text{sign}(\nabla_{\mathbf{x}} \mathcal{L}(\mathbf{x}^{(t)})))$$

where $\Pi_{\mathcal{B}_\infty(\mathbf{x}, \epsilon)}(\mathbf{z}) = \mathbf{x} + \text{clip}(\mathbf{z} - \mathbf{x}, -\epsilon, \epsilon)$ is projection onto the ℓ_∞ ball, and $\alpha < \epsilon$ is the step size.

PGD finds a strong local maximum of $\mathcal{L}$ inside the constraint set. With enough steps, it approximates the true maximum and is considered the gold standard for evaluating adversarial robustness.

Why PGD over FGSM? FGSM linearises the loss around $\mathbf{x}$, which is only accurate when ϵ is small. For larger ϵ , the true adversarial example lies far from the linear approximation. PGD with small steps α traverses the actual loss surface, finding higher-loss perturbations.

Projection operator. The projection onto the ℓ_∞ ball $\mathcal{B}_\infty(\mathbf{x}, \epsilon)$ is coordinate-wise clipping:

$$[\Pi_{\mathcal{B}_\infty(\mathbf{x}, \epsilon)}(\mathbf{z})]_i = \max(\mathbf{x}_i - \epsilon, \min(\mathbf{x}_i + \epsilon, z_i))$$

For ℓ_2 : $\Pi_{\mathcal{B}_2(\mathbf{x}, \epsilon)}(\mathbf{z}) = \mathbf{x} + \epsilon \cdot \frac{\mathbf{z} - \mathbf{x}}{\max(\|\mathbf{z} - \mathbf{x}\|_2, \epsilon)}$ (project onto sphere if outside, identity if inside).

Convergence of PGD. On a locally concave loss (satisfied near decision boundaries), PGD with k steps and step size $\alpha \leq \epsilon/k$ converges to the global maximum within the ℓ_∞ ball at rate $O(1/\sqrt{k})$. In practice, $k = 7$ steps with $\alpha = \epsilon/4$ is sufficient for training, while $k = 50$ is standard for evaluation.

EXAMPLE — PGD-7 Trace on CIFAR-10 Pixel

Start: $x_0 = 0.5, \epsilon = 8/255 \approx 0.031, \alpha = 2/255$.

Step 0: $x^{(0)} = x_0 + \text{Uniform}(-\epsilon, \epsilon) = 0.5 + 0.01 = 0.51$.

Step 1: $\nabla \mathcal{L} = 0.8 > 0$, step: $x^{(1)} = \text{clip}_{[0.469, 0.531]}(0.51 + 2/255) = 0.518$.

⋮

Step 7: Converges to $x^{(7)} = 0.531 = x_0 + \epsilon$ (boundary of ball, since loss gradient is consistently positive).

This single pixel example illustrates that PGD quickly reaches the constraint boundary in directions where the gradient consistently points outward.

Adversarial training with PGD. Madry et al. (2017) show that training on PGD adversarial examples with $k = 7$ steps provides robustness against *any* first-order attack (any attack

that uses only gradient information). This is because PGD with $k = 7$ already finds near-optimal perturbations; additional PGD steps or more sophisticated attacks provide minimal additional strength.

Auto-PGD (APGD). Croce & Hein (2020) improve PGD with adaptive step size scheduling: the step size is reduced when the attack fails to make progress, and the best-found perturbation is tracked:

$$\alpha_{t+1} = \begin{cases} \alpha_t/2 & \text{if no improvement in last } w \text{ steps} \\ \alpha_t & \text{otherwise} \end{cases}$$

APGD is now the standard component of the AutoAttack benchmark, which combines APGD, square attack (black-box), and FAB attack to provide a comprehensive robustness evaluation.

43.4 The Landscape of Adversarial Training

Adversarial training minimises the min-max objective $\min_{\theta} \mathbb{E}[\max_{\|\delta\| \leq \epsilon} \mathcal{L}(\theta, x + \delta, y)]$. Understanding its optimisation landscape is key to stable and effective training.

Inner maximisation difficulty. The inner max $\max_{\|\square\| \leq \epsilon} \mathcal{L}(f_{\theta}(\mathbf{x} + \square), y)$ is non-convex in $\square$ for a neural network f_{θ} . Multiple local maxima can exist, and PGD finds only a local one. However, for “strongly concave” loss landscapes near $\mathbf{x}$ (which holds for well-trained classifiers), PGD converges to a global maximum of the inner problem.

Non-convergence and oscillation. Naive alternating gradient updates for min-max can cycle rather than converge. The fundamental issue: the game is non-cooperative (two-player), and the Nash equilibrium may not be reachable by gradient dynamics. Remedies:

- *Multiple inner steps:* use $k = 7$ –10 PGD steps per gradient update of θ to keep the inner maximiser near-optimal.
- *Warm-starting:* initialise PGD from the previous iteration’s adversarial example, reducing the number of steps needed to find a strong attack.
- *Free adversarial training* (Shafahi et al., 2019): replay the same adversarial example for m epochs of the outer loop, amortising the cost of PGD to m times cheaper than standard AT.

The AT loss surface. The adversarial training loss $\mathcal{L}_{\text{AT}}(\theta) = \mathbb{E}[\max_{\delta} \mathcal{L}(\theta, \mathbf{x} + \delta)]$ is more complex than the standard training loss:

- More rugged (higher Hessian eigenvalues): the worst-case perturbation changes sharply as θ varies.
- Flatter global minima: the adversarially robust classifier must have large “safety margins” at all training examples, which enforces smoother decision boundaries and hence flatter loss basins.
- Harder to optimise: the effective condition number of $\nabla^2 \mathcal{L}_{\text{AT}}$ is typically 5–10 $\times$ larger than for standard training.

Adversarial Accuracy and Certified Robustness Comparison

Defence method	Clean Acc	PGD-20 Acc	Certified Acc
Standard training	94%	~0%	0%
AT (PGD-7, $\epsilon = 8/255$)	84%	51%	0%
AT + SmoothAdv	75%	55%	38%*
Randomised smoothing ($\sigma = 0.5$)	67%	—	49%

*Certified for ℓ_2 perturbations $\|\delta\|_2 \leq 0.5$; PGD-20 measures ℓ_∞ robustness. The two guarantees are not directly comparable.

Feature purification view. Adversarial training can be interpreted as learning to use only *robust features* (features that remain consistent under small perturbations) while discarding *non-robust features* (brittle texture correlations). This explains the clean accuracy drop: non-robust features are genuinely predictive for in-distribution data, but relying on them sacrifices robustness. The fundamental tradeoff is:

$$\text{clean accuracy} + \text{robustness} \leq C(d, n, \epsilon)$$

where C is a dataset- and architecture-specific constant. Tsipras et al. (2019) prove this tradeoff is information-theoretic for certain data distributions.

TRADES regularisation. Zhang et al. (2019) propose TRADES, which explicitly balances the accuracy-robustness tradeoff:

$$\min_{\theta} \mathbb{E} \left[\mathcal{L}(f_{\theta}(\mathbf{x}), y) + \frac{1}{\lambda} \max_{\|\square\| \leq \epsilon} \text{KL}(f_{\theta}(\mathbf{x}) \| f_{\theta}(\mathbf{x} + \square)) \right]$$

The first term is the standard cross-entropy loss (maximising clean accuracy). The second term penalises the KL divergence between clean and adversarial predictions (maximising robustness). The hyperparameter λ controls the tradeoff: $\lambda = 1$ gives Madry AT, larger λ emphasises clean accuracy.

EXAMPLE — TRADES vs. Madry AT on CIFAR-10 (WideResNet-34-10)

Method	λ	Clean Acc	AutoAttack Acc
Madry AT	—	84.9%	49.1%
TRADES	6	84.9%	56.6%
TRADES	3	86.3%	55.5%
TRADES	1	87.1%	53.2%

TRADES with $\lambda = 6$ achieves the highest robust accuracy (56.6% vs. 49.1%) at no clean accuracy cost. The key: the KL regularisation in TRADES penalises large local Lipschitz constants near training examples, directly optimising for certifiable robustness.

43.5 Black-Box Attacks and Score-Based Adversarial Examples

White-box attacks (FGSM, PGD) require access to model gradients. Black-box attacks operate without gradient access, querying only the model's outputs — a more realistic threat model for deployed systems.

Score-based vs. decision-based attacks.

- *Score-based*: attacker observes the full probability vector $f(\mathbf{x}) \in \Delta^C$ at each query.

- *Decision-based*: attacker observes only the hard label $\hat{y} = \arg \max_c f_c(\mathbf{x})$ at each query.

Decision-based attacks are harder (less information per query) but match real-world deployments (APIs often return only top- k labels, not full logits).

Natural Evolution Strategies (NES) attack. Ilyas et al. (2018) estimate the gradient via NES:

$$\nabla_{\mathbf{x}} \mathcal{L} \approx \frac{1}{n\sigma} \sum_{i=1}^n \mathcal{L}(\mathbf{x} + \sigma \square_i) \square_i, \quad \square_i \sim \mathcal{N}(0, \mathbf{I})$$

Each gradient estimate uses $2n$ queries (one for $\pm \square_i$). This estimate is unbiased:

$$\mathbb{E} \left[\frac{\mathcal{L}(\mathbf{x} + \sigma \square)}{\sigma} \right] = \frac{1}{\sigma} \mathbb{E}[\mathcal{L}(\mathbf{x} + \sigma \square)] = \nabla_{\mathbf{x}} \mathbb{E}_{\square}[\mathcal{L}(\mathbf{x} + \sigma \square)] \approx \nabla_{\mathbf{x}} \mathcal{L}(\mathbf{x})$$

The approximation error is $O(\sigma)$ (from the Taylor expansion). With $n = 50$ samples and $\sigma = 0.001$, a single NES gradient estimate requires 100 queries.

ZOO: Zeroth Order Optimisation. Chen et al. (2017) use coordinate-wise finite differences:

$$\frac{\partial \mathcal{L}}{\partial x_i} \approx \frac{\mathcal{L}(\mathbf{x} + h\mathbf{e}_i) - \mathcal{L}(\mathbf{x} - h\mathbf{e}_i)}{2h}$$

with step size h (analogous to PGD step size). Full gradient estimation: $2d$ queries per step. For $d = 224 \times 224 \times 3 \approx 150000$: infeasible directly. ZOO uses importance sampling (query only the top- k coordinates predicted to be important) and coordinate descent (update one coordinate at a time).

Square Attack. Andriushchenko et al. (2020) propose a score-based attack that operates by randomly selecting a square region of the image and optimising the perturbation within it:

1. Initialise $\square$ randomly in the ℓ_{∞} ball.
2. At each step: sample a random position (i, j) and size s ; propose a new perturbation in the $s \times s$ square.
3. Accept if $\mathcal{L}$ increases (hill-climbing); reject otherwise.

Square Attack achieves $\sim 5\text{--}10\times$ fewer queries than NES/ZOO for the same success rate, by exploiting the spatial structure of images (adversarial perturbations tend to be locally coherent).

Query Efficiency Comparison (ImageNet, $\ell_{\infty} \epsilon = 4/255$)

Attack	Attack Success	Queries (median)	Queries (max)
NES	95%	1 800	10 000
ZOO (importance)	95%	720	5 000
Square Attack	95%	210	1 000
Bandits (Prior)	95%	180	800

Bandits with data-dependent priors (Ilyas et al., 2019) uses transfer gradients from a surrogate model as a prior over which coordinates are most sensitive, achieving the lowest query count. Square Attack is architecture-agnostic and requires no surrogate model.

Hard-label attacks. When only the decision boundary is observable, Brendel et al. (2018) propose boundary attack: start from a large adversarial perturbation and walk along the

decision boundary to reduce $\|\square\|_2$ while remaining adversarial. The walk uses an isotropic random walk projected onto the decision boundary. HopSkipJump (Chen et al., 2020) improves efficiency by using binary search along the boundary direction, achieving $\sim 100\times$ speedup over boundary attack.

Certified Robustness: Randomised Smoothing

Randomised smoothing constructs a provably robust classifier by convolving the base model with Gaussian noise, yielding rigorous ℓ_2 certificates without assumptions about the network architecture.

44.1 The Smoothed Classifier

Randomised smoothing, introduced by Cohen, Rosenfeld & Kolter (2019), is the first approach to provide *certified* adversarial robustness at ImageNet scale. Rather than directly certifying the base classifier f , it certifies a *smoothed* version g that averages f over Gaussian noise.

Definition. Given a base classifier $f : \mathbb{R}^d \rightarrow \mathcal{Y}$ and noise level $\sigma > 0$, the smoothed classifier is:

$$g(\mathbf{x}) = \arg \max_{c \in \mathcal{Y}} P_{\square \sim \mathcal{N}(0, \sigma^2 \mathbf{I})} [f(\mathbf{x} + \square) = c]$$

g predicts the class that f most frequently predicts when the input is corrupted by Gaussian noise.

The neyman-Pearson connection. The key insight is that the smoothed classifier g has a simple decision region: by the Neyman-Pearson lemma, the most powerful test for distinguishing $\mathcal{N}(\mathbf{x}, \sigma^2 \mathbf{I})$ from $\mathcal{N}(\mathbf{x}', \sigma^2 \mathbf{I})$ is a halfspace. Consequently:

$$g(\mathbf{x}) = c_A \quad \Rightarrow \quad g(\mathbf{x}') = c_A \quad \forall \|\mathbf{x}' - \mathbf{x}\|_2 \leq R$$

where R depends on how dominant class c_A is. This gives an ℓ_2 certified radius.

Certified Prediction

Let $\underline{p}_A$ and $\bar{p}_B$ be lower and upper bounds (with confidence $1 - \alpha$) on:

$$p_A = P[f(\mathbf{x} + \square) = c_A], \quad p_B = \max_{c \neq c_A} P[f(\mathbf{x} + \square) = c]$$

If $\underline{p}_A > \bar{p}_B$, then g is certified to predict c_A within ℓ_2 radius:

$$R = \frac{\sigma}{2} (\Phi^{-1}(\underline{p}_A) - \Phi^{-1}(\bar{p}_B))$$

where Φ^{-1} is the inverse standard normal CDF.

Abstention rule. If $\underline{p}_A \leq 0.5$, the smoothed classifier cannot certify any prediction and should *abstain*. This happens when f is highly uncertain under noise — typically near decision boundaries or for inputs far from the training distribution.

Training for smoothing. The base classifier f should be trained with *noise augmentation* to perform well on noisy inputs:

$$\mathcal{L}_{\text{smooth}} = \mathbb{E}_{\square \sim \mathcal{N}(0, \sigma^2 \mathbf{I})} [\ell(f(\mathbf{x} + \square), y)]$$

This is equivalent to training a classifier on a Gaussian-blurred version of the data, which encourages f to predict consistently across the noise neighbourhood. Without noise augmentation, the base classifier quickly becomes inaccurate at large σ , limiting the achievable certified radius.

EXAMPLE — Certified Radius on ImageNet ($\sigma = 0.5$)

On ResNet-50 with Gaussian noise augmentation:

- Certified accuracy at ℓ_2 radius 0.5: 49%.
- Certified accuracy at ℓ_2 radius 1.0: 37%.
- Certified accuracy at ℓ_2 radius 2.0: 20%.

Compare to a standard ResNet-50: 0% certified accuracy at any positive radius (no certificate). Randomised smoothing achieves the first non-trivial ℓ_2 certificates for ImageNet.

Given a base classifier $f : \mathcal{X} \rightarrow \mathcal{Y}$, define the smoothed classifier g as:

$$g(\mathbf{x}) = \arg \max_{c \in \mathcal{Y}} P_{\square \sim \mathcal{N}(\mathbf{0}, \sigma^2 \mathbf{I})} [f(\mathbf{x} + \square) = c]$$

$g(\mathbf{x})$ returns the class that f predicts most often under Gaussian noise.

44.2 Certification Theorem

Cohen et al. (2019) Certificate

Suppose $g(\mathbf{x}) = c_A$ and the probability

$$p_A = P_{\square} [f(\mathbf{x} + \square) = c_A] \geq \underline{p}_A > \frac{1}{2}$$

and let $p_B = \max_{c \neq c_A} P_{\square} [f(\mathbf{x} + \square) = c] \leq \bar{p}_B$.

Then $g(\mathbf{x}') = c_A$ for all $\mathbf{x}'$ with:

$$\|\mathbf{x}' - \mathbf{x}\|_2 < R = \frac{\sigma}{2} (\Phi^{-1}(\underline{p}_A) - \Phi^{-1}(\bar{p}_B))$$

where Φ^{-1} is the inverse standard normal CDF.

Proof sketch. Neyman-Pearson lemma shows that the hardest perturbation to defend against within an ℓ_2 ball is a translation of the Gaussian distribution. The decision boundary between c_A and any other class c under additive Gaussian noise corresponds to a halfspace in the frequency domain, and the probability gap $\underline{p}_A - \bar{p}_B$ maps to a geometric margin via Φ^{-1} .

EXAMPLE — Computing a certificate

$\sigma = 0.25, \underline{p}_A = 0.90, \bar{p}_B = 0.05$.

$$\Phi^{-1}(0.90) \approx 1.282, \quad \Phi^{-1}(0.05) \approx -1.645$$

$$R = \frac{0.25}{2} (1.282 - (-1.645)) = 0.125 \times 2.927 \approx 0.366$$

The smoothed classifier is certified correct for any ℓ_2 perturbation of radius up to 0.366.

44.3 Practical Certification via Monte Carlo

Exact computation of $p_A = \Pr[f(\mathbf{x} + \square) = c_A]$ requires integrating over all Gaussian noise realisations — intractable for a deep network. Cohen et al. (2019) use Monte Carlo estimation

with Clopper-Pearson confidence intervals.

Algorithm. Given input $\mathbf{x}$, class c_A , confidence $1 - \alpha$, and n_{samp} Monte Carlo samples:

1. Sample $\square^{(1)}, \dots, \square^{(n_{\text{samp}})} \sim \mathcal{N}(0, \sigma^2 \mathbf{I})$.
2. Count: $k = |\{i : f(\mathbf{x} + \square^{(i)}) = c_A\}|$.
3. Compute lower bound: $\underline{p}_A = \text{BinomCI}(k, n_{\text{samp}}, \alpha/2)_{\text{lower}}$ (Clopper-Pearson).
4. If $\underline{p}_A > 0.5$: certify with radius $R = \sigma \Phi^{-1}(\underline{p}_A)$.

The Clopper-Pearson interval gives a one-sided lower bound: $\Pr[\hat{p}_A \geq \underline{p}_A] \geq 1 - \alpha/2$.

Certified accuracy at radius r . A test point is “certified at r ” if: (1) the smoothed classifier predicts correctly, and (2) the certified radius $R \geq r$. Certified accuracy at radius r = fraction of test points certified at r .

EXAMPLE — Certification Statistics for Cohen et al. (2019) on ImageNet

Model: ResNet-110 trained with Gaussian noise augmentation $\sigma = 0.5$.

Test: $n_{\text{samp}} = 1000$, $\alpha = 0.001$.

Radius r (pixels, ℓ_2)	Certified Accuracy
0 (clean, abstain-free)	49%
0.5	38%
1.0	28%
1.5	19%
2.0	12%

At $r = 0.5$ (ℓ_2), 38% of ImageNet test images are classified correctly *and* certified robust. No adversarial attack with $\|\square\|_2 \leq 0.5$ can fool the smoothed classifier on these images — guaranteed by the proof, not just empirically.

Tight certification. The Cohen et al. bound is tight in the following sense: for any $p_A > 1/2$ and any function f , there exists an adversarial perturbation of ℓ_2 norm exactly $R = \sigma \Phi^{-1}(p_A)$ that changes the smoothed classifier’s prediction. The certificate is therefore the emphbest possible ℓ_2 guarantee achievable by randomised smoothing.

In practice, $\underline{p}_A$ is estimated by sampling $n = 10^5$ Gaussian noises and counting how often f predicts c_A . A one-sided Clopper-Pearson confidence interval gives $\underline{p}_A$ with probability $1 - \alpha$. The certification is then valid with probability $1 - \alpha$ over the randomness of the noise samples.

44.4 Certified Training and the SmoothAdv Objective

Randomised smoothing provides certification at test time, but the base classifier f must be trained to be robust to Gaussian noise. Standard training with noise augmentation is suboptimal; certified training directly optimises the certification radius.

SmoothAdv training. Salman et al. (2019) propose training on adversarial examples generated against the smoothed classifier. The training objective:

$$\min_{\theta} \mathbb{E}_{(\mathbf{x}, y)} \max_{\|\square\|_2 \leq \rho} \mathbb{E}_{\square \sim \mathcal{N}(0, \sigma^2 \mathbf{I})} \mathcal{L}(f_{\theta}(\mathbf{x} + \square + \square), y)$$

The inner maximisation finds perturbations that are adversarial even after Gaussian smooth-

ing. This is harder than standard adversarial training: the adversary must account for the Gaussian averaging.

MACER training. Zhai et al. (2020) directly maximise the certification radius during training. The loss includes a term:

$$\mathcal{L}_{\text{MACER}} = \mathcal{L}_{\text{CE}} + \lambda \cdot \max(0, \gamma - \sigma \Phi^{-1}(p_A(\theta)))$$

where $p_A(\theta)$ is the softmax probability of the correct class, used as a surrogate for the certification probability. This directly pushes p_A above 0.5, increasing the certified radius.

EXAMPLE — Certified Accuracy Comparison on CIFAR-10 ($\sigma = 0.5$)

Training Method	Clean Acc	Certified Acc ($r = 0.5$)
Gaussian augmentation	74%	49%
SmoothAdv	67%	54%
MACER	71%	59%

MACER improves certified accuracy by 10% over standard Gaussian training while sacrificing only 3% clean accuracy. The direct optimisation of the certification metric provides a substantial benefit over indirect methods.

44.5 Interval Bound Propagation and Complete Verification

Randomised smoothing provides statistical certificates; deterministic certified robustness methods provide *exact* worst-case guarantees by propagating bounds through the network.

Interval Bound Propagation (IBP). IBP (Gowal et al., 2018) propagates intervals $[\underline{x}_i, \bar{x}_i]$ through each layer, bounding the network output over all inputs in the ℓ_∞ ball $\|\square\|_\infty \leq \epsilon$.

For an affine layer $\mathbf{z} = \mathbf{W}\mathbf{x} + \mathbf{b}$:

$$\begin{aligned} \underline{z}_j &= \sum_i [\mathbf{W}_{ji}]^+ \underline{x}_i + [\mathbf{W}_{ji}]^- \bar{x}_i + b_j \\ \bar{z}_j &= \sum_i [\mathbf{W}_{ji}]^+ \bar{x}_i + [\mathbf{W}_{ji}]^- \underline{x}_i + b_j \end{aligned}$$

where $[a]^+ = \max(a, 0)$ and $[a]^- = \min(a, 0)$.

For a ReLU activation $\mathbf{a} = \text{ReLU}(\mathbf{z})$:

$$\underline{a}_i = \max(\underline{z}_i, 0), \quad \bar{a}_i = \max(\bar{z}_i, 0)$$

IBP for certification. Given input $\mathbf{x}$ and true class c , the network is certified robust at perturbation radius ϵ if the lower bound on the logit difference $\underline{f}_c(\mathbf{x}') - \overline{f}_j(\mathbf{x}') > 0$ for all $j \neq c$ and all $\mathbf{x}'$ with $\|\mathbf{x}' - \mathbf{x}\|_\infty \leq \epsilon$.

IBP training. Models trained with standard loss are not certifiable (tight intervals are impossible). IBP training minimises:

$$\mathcal{L}_{\text{IBP}} = \kappa \mathcal{L}_{\text{CE}}(f(\mathbf{x}), y) + (1 - \kappa) \mathcal{L}_{\text{CE}}\left(\frac{f(\mathbf{x}') + \overline{f}(\mathbf{x}')}{2}, y\right)$$

The second term trains on the *worst-case* interval midpoint, forcing the network to produce small intervals. The hyperparameter κ is annealed from 1 to 0 during training.

Limitations of IBP. IBP is an over-approximation: the actual reachable set of network outputs over $\mathcal{B}_\infty(\mathbf{x}, \epsilon)$ is contained within but may be much smaller than the IBP interval. For deep networks, IBP bounds can be exponentially loose. More precise methods:

Verification Methods Comparison			
Method	Completeness	Cost	Tightness
IBP	Incomplete (over-approx)	$O(dL)$	Loose for deep nets
CROWN	Incomplete (linear relaxation)	$O(d^2L)$	Tighter than IBP
α -CROWN	Incomplete (optimised)	$O(d^2L)$ per iter	Very tight
MIP (Mixed Integer Program)	Complete	NP-hard	Exact

CROWN: linear relaxation. Zhang et al. (2018) propose CROWN, which uses linear relaxations of each neuron:

- For a neuron with pre-activation bounds $[z, \bar{z}]$:
- If $\bar{z} \leq 0$ (always inactive): $\hat{a} = 0$.
- If $\underline{z} \geq 0$ (always active): $\hat{a} = z$.
- Otherwise (unstable): use linear lower/upper bounds:

$$\frac{\bar{z}}{\bar{z} - \underline{z}}(z - \underline{z}) \leq \text{ReLU}(z) \leq \frac{\bar{z}}{\bar{z} - \underline{z}}z \quad (\text{triangle relaxation})$$

Propagating these linear bounds through the network gives a tighter certificate than IBP, at the cost of quadratic (in neuron count) rather than linear computation.

EXAMPLE — IBP vs. CROWN Certified Accuracy on CIFAR-10

Method	Train method	Certified Acc ($\epsilon = 2/255$)
IBP	Standard	0%
IBP	IBP training	48.9%
CROWN-IBP	CROWN-IBP training	54.0%
α -CROWN	Mixed training	58.3%
Randomised Smoothing ($\sigma = 0.12$)	Gaussian aug.	61%*

*Randomised Smoothing certifies ℓ_2 balls; the ℓ_∞ comparison is approximate. IBP-trained models achieve higher certified accuracy than standard models under ℓ_∞ perturbations. The CROWN family gives tighter certificates at higher computational cost. Randomised smoothing provides probabilistic (not deterministic) certificates but scales better to larger networks.

Differential Privacy and Federated Learning

Differential privacy formalises the privacy guarantee of randomised algorithms. We derive the Gaussian mechanism, the moments accountant, and show how DP-SGD and federated averaging interact with privacy budgets.

45.1 Differential Privacy

Differential privacy (Dwork et al., 2006) is the gold-standard mathematical framework for privacy-preserving computation. It provides a rigorous, quantitative guarantee that the output of an algorithm reveals “almost nothing” about any single individual’s data.

Motivation. Training a model on sensitive data (medical records, financial transactions) may leak individual data points through model weights. Membership inference attacks can determine with high probability whether a specific record was in the training set. DP prevents such leakage with provable guarantees.

(ϵ, δ) -Differential Privacy

A randomised mechanism $\mathcal{M} : \mathcal{D}^n \rightarrow \mathcal{O}$ is (ϵ, δ) -differentially private if for all pairs of datasets D, D' differing in at most one entry, and all measurable output sets $S \subseteq \mathcal{O}$:

$$\Pr[\mathcal{M}(D) \in S] \leq e^\epsilon \cdot \Pr[\mathcal{M}(D') \in S] + \delta$$

Intuition. If ϵ is small, the output distributions on any two neighbouring datasets are almost identical. An adversary seeing the output cannot determine whether a specific individual was in the dataset. The parameter δ allows a small probability of failure: with probability $1 - \delta$, the e^ϵ bound holds.

Typical parameter values.

- $\epsilon \leq 1$: strong privacy (the log-ratio of probabilities is small).
- $\epsilon = 8$ – 10 : commonly accepted in practice for large n (e.g., US Census uses $\epsilon = 19.61$).
- $\delta \leq 1/n^2$: negligible failure probability (should be much smaller than $1/n$).

Sensitivity. The ℓ_2 -sensitivity of a function $f : \mathcal{D}^n \rightarrow \mathbb{R}^d$ is:

$$\Delta_2(f) = \max_{\text{neighbours } D, D'} \|f(D) - f(D')\|_2$$

For gradient computation on individual examples: if the per-sample gradient is clipped to norm C , then $\Delta_2(\text{gradient}) \leq 2C$ (changing one sample changes the gradient by at most its two clipped contributions).

EXAMPLE — Sensitivity of the Mean Gradient

Dataset $D = \{z_1, \dots, z_n\}$, function $f(D) = \frac{1}{n} \sum_{i=1}^n \ell(\theta; z_i)$. Gradient:

$$\nabla f(D) = \frac{1}{n} \sum_{i=1}^n \nabla_{\theta} \ell(\theta; z_i)$$

If individual gradients are clipped to $\|\nabla_{\theta}\ell(\theta; z_i)\|_2 \leq C$, then replacing z_j by z'_j :

$$\|\nabla f(D) - \nabla f(D')\|_2 = \frac{1}{n} \|\nabla_{\theta}\ell(\theta; z_j) - \nabla_{\theta}\ell(\theta; z'_j)\|_2 \leq \frac{2C}{n}$$

So $\Delta_2(\nabla f) = 2C/n$.

Differential privacy (DP), introduced by Dwork et al. (2006), is a rigorous mathematical framework for quantifying and bounding the privacy leakage of algorithms that operate on sensitive data. It has become the standard privacy guarantee in machine learning, adopted by organisations including Apple, Google, and the US Census Bureau.

Motivation. Consider a medical database where each record is a patient's health data. A statistical query (e.g., "what fraction of patients have diabetes?") might leak information about individuals, especially in sparse populations. DP provides a worst-case guarantee: regardless of what an adversary knows about $n - 1$ individuals, they learn almost nothing about the n -th individual from the query output.

(ϵ, δ) -Differential Privacy

A randomised mechanism $\mathcal{M} : \mathcal{D} \rightarrow \mathcal{R}$ is (ϵ, δ) -differentially private if for all adjacent datasets S, S' (differing in exactly one record) and all measurable output sets $O \subseteq \mathcal{R}$:

$$P[\mathcal{M}(S) \in O] \leq e^{\epsilon} \cdot P[\mathcal{M}(S') \in O] + \delta$$

- ϵ (privacy budget): smaller ϵ = stronger privacy. $\epsilon \leq 1$ is "strong", $\epsilon \leq 10$ is "moderate".
- δ (failure probability): probability of catastrophic privacy failure. Should be $\ll 1/n$.
- Pure DP: $\delta = 0$ (exponential mechanism, Laplace mechanism).
- Approximate DP: $\delta > 0$ allows the stronger Gaussian mechanism.

The sensitivity framework. To add calibrated noise, we need to know how much a single record can affect a query. The ℓ_2 -sensitivity of $f : \mathcal{D} \rightarrow \mathbb{R}^k$ is:

$$\Delta_2(f) = \max_{S \sim S'} \|f(S) - f(S')\|_2$$

The Gaussian mechanism adds noise $\mathcal{N}(0, \sigma^2 \mathbf{I}_k)$ with $\sigma = \frac{\Delta_2(f) \sqrt{2 \ln(1.25/\delta)}}{\epsilon}$ to achieve (ϵ, δ) -DP.

Composition theorems. A key property of DP is graceful composition: if $\mathcal{M}_1$ is (ϵ_1, δ_1) -DP and $\mathcal{M}_2$ is (ϵ_2, δ_2) -DP, then their sequential composition is $(\epsilon_1 + \epsilon_2, \delta_1 + \delta_2)$ -DP. More precisely, advanced composition gives $(\sqrt{2k \ln(1/\delta')}(\epsilon) + k\epsilon(e^{\epsilon} - 1), k\delta + \delta')$ -DP for k compositions.

Privacy amplification by subsampling. If $\mathcal{M}$ is (ϵ, δ) -DP, then the mechanism that runs $\mathcal{M}$ on a uniformly random subsample of size m from n records is $(\log(1 + \frac{m}{n}(e^{\epsilon} - 1)), \frac{m\delta}{n})$ -DP. This amplification is crucial for DP-SGD (§45.3), where each mini-batch is a random subsample.

EXAMPLE — Privacy Budget in Practice

Training a language model with DP-SGD for $T = 10^5$ steps, batch size $B = 256$, dataset size $n = 10^6$, noise multiplier $z = 1.1$:

- Sampling rate: $q = B/n = 2.56 \times 10^{-4}$.
- Using RDP accountant: $\epsilon \approx 8.0$ at $\delta = 10^{-6}$.
- Reducing noise ($z = 0.7$): $\epsilon \approx 200$ — essentially no privacy guarantee.

The trade-off between privacy (ϵ) and model quality is the central challenge of DP machine learning.

(ϵ, δ) -Differential Privacy

A randomised mechanism $\mathcal{M} : \mathcal{D} \rightarrow \mathcal{R}$ is (ϵ, δ) -differentially private if for all neighbouring datasets D, D' (differing in one record) and all measurable sets $S \subseteq \mathcal{R}$:

$$P[\mathcal{M}(D) \in S] \leq e^\epsilon \cdot P[\mathcal{M}(D') \in S] + \delta$$

ϵ is the privacy budget (smaller = more private); δ is a failure probability (typically $\ll 1/n$).

45.2 The Gaussian Mechanism

The Gaussian mechanism adds calibrated Gaussian noise to achieve (ϵ, δ) -DP. Unlike the Laplace mechanism (which achieves $(\epsilon, 0)$ -DP but requires ℓ_1 sensitivity), the Gaussian mechanism uses ℓ_2 sensitivity and is more natural for gradient perturbation.

Gaussian Mechanism

Given a function $f : \mathcal{D}^n \rightarrow \mathbb{R}^d$ with ℓ_2 -sensitivity Δ_2 , the Gaussian mechanism:

$$\mathcal{M}_G(D) = f(D) + \mathcal{N}(\mathbf{0}, \sigma^2 \mathbf{I}_d)$$

is (ϵ, δ) -DP for any $\epsilon \in (0, 1)$ if:

$$\sigma \geq \frac{\Delta_2 \sqrt{2 \ln(1.25/\delta)}}{\epsilon}$$

Proof sketch. We need to show: for any neighbouring D, D' and any $S \subseteq \mathbb{R}^d$:

$$\Pr[\mathcal{M}_G(D) \in S] \leq e^\epsilon \cdot \Pr[\mathcal{M}_G(D') \in S] + \delta$$

Let $\Delta = f(D) - f(D')$, $\|\Delta\|_2 \leq \Delta_2$. The ratio of densities:

$$\frac{p_D(\mathbf{z})}{p_{D'}(\mathbf{z})} = \exp\left(\frac{\mathbf{z}^\top \Delta - \|\Delta\|^2/2}{\sigma^2}\right)$$

This ratio exceeds e^ϵ when $\mathbf{z}^\top \Delta / \sigma^2 > \epsilon + \|\Delta\|^2 / (2\sigma^2)$, which happens with probability at most:

$$\Pr\left[\mathcal{N}(0, 1) > \frac{\epsilon\sigma}{\Delta_2} - \frac{\Delta_2}{2\sigma}\right] \leq \Phi\left(-\frac{\epsilon\sigma}{\Delta_2}\right) \leq \delta$$

The last inequality requires $\epsilon\sigma/\Delta_2 \geq \sqrt{2 \ln(1.25/\delta)}$, which gives the stated σ requirement.

EXAMPLE — Noise Level for a Gradient with $\Delta_2 = 2C$

Gradient clipping norm $C = 1$, so $\Delta_2 = 2$ (as derived in §45.1). Privacy parameters $\epsilon = 1, \delta = 10^{-5}$.

Required noise standard deviation:

$$\sigma \geq \frac{2\sqrt{2\ln(1.25/10^{-5})}}{1} = 2\sqrt{2\ln(125000)} \approx 2\sqrt{23.5} \approx 9.7$$

So each gradient coordinate receives noise $\mathcal{N}(0, 9.7^2)$. For a gradient of magnitude $\sim C/n = 1/n$, the signal-to-noise ratio is $1/(9.7n)$, which is extremely small for any practical batch. This is why DP training with small ε incurs significant utility loss: the required noise overwhelms the gradient signal for individual examples.

With mini-batching and the moments accountant, this noise level is accumulated over many steps, allowing the model to learn despite per-step $\text{SNR} < 1$.

For a query $q : \mathcal{D} \rightarrow \mathbb{R}^d$ with ℓ_2 sensitivity $\Delta_2 = \max_{D, D'} \|q(D) - q(D')\|_2$, the Gaussian mechanism adds noise:

$$\mathcal{M}(D) = q(D) + \mathcal{N}(\mathbf{0}, \sigma^2 \mathbf{I}_d)$$

Gaussian Mechanism Privacy Guarantee

For $\varepsilon < 1$, the Gaussian mechanism with $\sigma \geq \frac{\Delta_2 \sqrt{2\ln(1.25/\delta)}}{\varepsilon}$ is (ε, δ) -DP.

Derivation. The log-likelihood ratio $\ln \frac{P[\mathcal{M}(D) \in S]}{P[\mathcal{M}(D') \in S]}$ for any S is bounded by the maximum over all $\mathbf{z} \in \mathbb{R}^d$ of:

$$\ln \frac{\phi_\sigma(\mathbf{z} - q(D))}{\phi_\sigma(\mathbf{z} - q(D'))} = \frac{1}{2\sigma^2} (\|\mathbf{z} - q(D')\|^2 - \|\mathbf{z} - q(D)\|^2)$$

In the worst case (when $\mathbf{z}$ is in the direction of $q(D) - q(D')$), this is $\frac{\Delta_2^2 + 2\Delta_2 \|\mathbf{z} - q(D)\|}{2\sigma^2}$; integrating over the Gaussian tail for the δ part gives the stated condition.

45.3 DP-SGD and the Moments Accountant

DP-SGD clips per-sample gradients and adds Gaussian noise at each step:

$$\mathbf{g}_i \leftarrow \mathbf{g}_i / \max\left(1, \frac{\|\mathbf{g}_i\|_2}{C}\right), \quad \tilde{\mathbf{g}} = \frac{1}{B} \left(\sum_i \mathbf{g}_i + \mathcal{N}(\mathbf{0}, \sigma^2 C^2 \mathbf{I}) \right)$$

The privacy cost accumulates over T steps. Naive composition gives $T\varepsilon$ total, but the moments accountant gives a tighter bound: for subsampling ratio $q = B/n$ and noise multiplier σ , the total (ε, δ) -DP budget after T steps is approximately:

$$\varepsilon \approx q\sqrt{2T\ln(1/\delta)}/\sigma \quad (\text{leading term})$$

45.3b Privacy Amplification by Subsampling

A key insight in DP-SGD is that mini-batch sampling provides “free” privacy amplification: running a DP mechanism on a random subsample of the data is more private than running it on the full dataset.

Amplification theorem (Balle et al., 2018). If $\mathcal{M}$ is $(\varepsilon_0, \delta_0)$ -DP and we apply it to a random subsample of size b from n records (sampling rate $q = b/n$):

$$\mathcal{M}_{\text{subsampled}} \text{ is } (O(q\varepsilon_0), q\delta_0 + \delta')\text{-DP}$$

For small ε_0 : the effective $\varepsilon \approx q\varepsilon_0 = (b/n)\varepsilon_0$.

DP-SGD privacy accounting. Each step of DP-SGD clips per-sample gradients to norm C and adds noise $\mathcal{N}(0, \sigma^2 C^2 \mathbf{I})$. Per-step privacy: $(\varepsilon_{\text{step}}, \delta_{\text{step}})$ with:

$$\varepsilon_{\text{step}} \approx \frac{\sqrt{2 \ln(1.25/\delta_{\text{step}})}}{\sigma}$$

After T steps with subsampling rate q , the moments accountant (Abadi et al., 2016) gives a tight cumulative bound:

$$\varepsilon_{\text{total}} = O\left(\frac{q\sqrt{T}}{\sigma}\right)$$

The $\sqrt{T}$ growth (vs. T for naive composition) is the key advantage of the moments accountant over basic composition.

EXAMPLE — DP-SGD Privacy Budget for MNIST Training

$n = 60,000$, batch $b = 256$ ($q = 256/60000 \approx 4.3 \times 10^{-3}$), noise $\sigma = 1.1$, clip $C = 1$, epochs = 60, $T = 60 \times 60000/256 = 14,062$ steps.

Moments accountant result: $\varepsilon = 9.9$ at $\delta = 10^{-5}$.

Naive basic composition: $\varepsilon = T \cdot q\varepsilon_{\text{step}} = 14062 \times 4.3 \times 10^{-3} \times 1.4 \approx 85$ (much looser).

With DP-SGD, the model achieves $\sim 97\%$ test accuracy (vs. 99% without privacy), a reasonable accuracy-privacy tradeoff at $\varepsilon \approx 10$.

45.4 Federated Averaging (FedAvg)

In federated learning, K clients each hold local data $\mathcal{D}_k$. FedAvg:

1. Server broadcasts $\square_t$ to a random subset S_t of clients.
2. Each client $k \in S_t$ runs τ local SGD steps to get $\square_t^{(k)}$.
3. Server aggregates: $\square_{t+1} = \sum_{k \in S_t} \frac{n_k}{n} \square_t^{(k)}$ where $n_k = |\mathcal{D}_k|$.

Combining DP with FedAvg. Each client applies DP-SGD locally; only the noisy aggregated update is sent to the server, protecting individual records. The central server never sees raw gradients.

45.5 Convergence Theory of Federated Averaging

Federated Averaging (FedAvg) is an approximation to distributed gradient descent, and its convergence depends on the degree of heterogeneity between client data distributions.

Convergence in the homogeneous case. When all clients have i.i.d. data ($p_k = p$ for all k), FedAvg with τ local steps is equivalent to mini-batch SGD with effective batch size $K\tau$ (where K is the number of clients per round). The convergence rate is the same as SGD:

$$\frac{1}{T} \sum_{t=1}^T \mathbb{E} \|\nabla \mathcal{L}(\theta_t)\|^2 \leq \frac{2(\mathcal{L}(\theta_0) - \mathcal{L}^*)}{\eta T} + \eta L \sigma^2$$

where L is the Lipschitz constant of $\nabla \mathcal{L}$ and σ^2 is the gradient variance.

The heterogeneity gap. When clients have non-i.i.d. data, local SGD introduces *client drift*: each client's local model drifts toward its own local optimum, and averaging drifted models may not approximate the global optimum.

Client Drift Bound (Li et al., 2020)

For K clients with local losses $\mathcal{L}_k$, global loss $\mathcal{L} = \frac{1}{K} \sum_k \mathcal{L}_k$, and heterogeneity measure $\Gamma = \mathcal{L}^* - \frac{1}{K} \sum_k \mathcal{L}_k^*$ (non-negative, zero when all local optima coincide):

$$\frac{1}{T} \sum_t \mathbb{E} \|\nabla \mathcal{L}(\theta_t)\|^2 \leq \frac{2(\mathcal{L}_0 - \mathcal{L}^*)}{\eta \tau T} + \underbrace{6L\Gamma + \frac{\sigma^2}{K}}_{\text{bias term from heterogeneity}}$$

The bias term $6L\Gamma$ does not decrease with T or K : *FedAvg does not converge to the global optimum with non-i.i.d. data.*

SCAFFOLD: correcting client drift. Karimireddy et al. (2020) propose SCAFFOLD (Stochastic Controlled Averaging for Federated Learning), which maintains control variates $\mathbf{c}_k$ at each client to correct for the drift:

$$\theta^{(k)} \leftarrow \theta^{(k)} - \eta(\nabla \mathcal{L}_k(\theta^{(k)}) - \mathbf{c}_k + \mathbf{c})$$

where $\mathbf{c} = \frac{1}{K} \sum_k \mathbf{c}_k$ is the global control variate and $\mathbf{c}_k$ is the client control variate. This zeroes out the client drift, restoring convergence to the global optimum at the same rate as distributed SGD.

Communication-computation tradeoff. In federated learning, communication is the bottleneck (not computation). The key metric is the number of communication rounds R to achieve ε -accuracy:

Algorithm	Rounds R	Per-round cost
Distributed SGD ($\tau = 1$)	$O(1/\varepsilon^2)$	$O(d)$ per client
FedAvg (τ local steps)	$O(1/(\varepsilon^2 \tau))$	$O(\tau d)$ per client
SCAFFOLD	$O(1/\varepsilon^2)$ (global minimum)	$O(\tau d) + \text{control variates}$

FedAvg reduces rounds by $1/\tau$ at the cost of convergence bias. SCAFFOLD corrects the bias but requires transmitting additional control variates (doubling per-round communication).

Personalised federated learning. When clients have very different data distributions, learning a single global model is suboptimal. Personalised FL algorithms find client-specific models θ_k that are close to a global model θ :

$$\min_{\theta, \{\theta_k\}} \frac{1}{K} \sum_k \left[\mathcal{L}_k(\theta_k) + \frac{\lambda}{2} \|\theta_k - \theta\|^2 \right]$$

This is the MAML meta-learning objective: θ is the global initialisation, and $\theta_k = \theta - \eta \nabla \mathcal{L}_k(\theta)$ is the personalised model after one gradient step. pFedMe (Dinh et al., 2020) solves this exactly using Moreau envelope proximal operators.

Domain Adaptation: DANN and Shift Bounds

We formalise domain shift, derive the Ben-David bound relating target error to source error and domain divergence, and analyse Domain-Adversarial Neural Networks (DANN).

46.1 Covariate Shift Formalisation

Domain adaptation addresses the problem of learning from a labelled *source* domain and deploying on an unlabelled *target* domain with a different input distribution.

Formal setup. Let $\mathcal{D}_S = (P_S(X), P(Y|X))$ be the source domain (labelled) and $\mathcal{D}_T = (P_T(X), P(Y|X))$ be the target domain (unlabelled). The key assumption:

- **Covariate shift:** $P_S(X) \neq P_T(X)$ but $P(Y|X)$ is shared. The marginal input distribution changes but the conditional label distribution is the same.
- **Full dataset shift:** both $P(X)$ and $P(Y|X)$ change. More general but harder.

Importance weighting. Under covariate shift, the optimal source-trained predictor may be suboptimal for the target. However, we can correct for the shift by importance weighting:

$$\mathbb{E}_{P_T}[\mathcal{L}(f(X), Y)] = \mathbb{E}_{P_S} \left[\frac{P_T(X)}{P_S(X)} \mathcal{L}(f(X), Y) \right]$$

So the target risk can be estimated from source data using importance weights $w(x) = P_T(x)/P_S(x)$.

Estimating importance weights. We can estimate $w(x)$ without knowing P_S and P_T explicitly. Train a binary classifier $D(x)$ to distinguish source ($d = 0$) from target ($d = 1$) samples:

$$\Pr[d = 1|x] = \frac{P_T(x)}{P_S(x) + P_T(x)} \implies w(x) = \frac{P_T(x)}{P_S(x)} = \frac{\Pr[d = 1|x]}{\Pr[d = 0|x]}$$

EXAMPLE — Covariate Shift in Medical Imaging

Source: X-rays from hospital A ($n_S = 10000$, labelled). Target: hospital B ($n_T = 500$, unlabelled). Distribution shift: hospital B uses a different scanner model with systematically lower contrast.

A domain classifier $D(x)$ trained to distinguish A vs. B images achieves 82% accuracy, indicating significant distributional shift.

Importance weights: images from hospital A that “look like” hospital B images receive $w(x) > 1$; images that are very hospital-A-specific receive $w(x) \approx 0$.

After importance-weighted training, target accuracy improves from 71% (unweighted) to 78% (weighted), recovering much of the distribution shift penalty.

Limitations of importance weighting. When P_S and P_T are very different (high distribution shift), importance weights become very large for some samples and near-zero for others, leading to high-variance estimates. The effective sample size is $n_{\text{eff}} = (\sum_i w_i)^2 / \sum_i w_i^2$; with extreme weights, $n_{\text{eff}} \ll n_S$. Feature-based domain adaptation (DANN, §46.3) avoids this by directly aligning the distributions in feature space.

Covariate shift is one of the most common forms of distribution shift in real-world machine learning: the marginal distribution of inputs $p(\mathbf{x})$ differs between training and test, but the conditional $p(y|\mathbf{x})$ remains the same.

Formal definition. Let $\mathcal{P}_s$ denote the source (training) distribution and $\mathcal{P}_t$ the target (test) distribution. Under covariate shift:

$$p_s(\mathbf{x}) \neq p_t(\mathbf{x}) \quad \text{but} \quad p_s(y|\mathbf{x}) = p_t(y|\mathbf{x}) \quad \forall \mathbf{x}$$

This assumption is plausible when the labelling mechanism is fixed (same task) but the data collection process changes — e.g., different demographics, different time periods, different sensors.

Importance Weighting Correction

The target population risk can be written as a reweighted source expectation:

$$R_t(h) = \mathbb{E}_{(\mathbf{x},y) \sim \mathcal{P}_t} [\ell(h(\mathbf{x}), y)] = \mathbb{E}_{(\mathbf{x},y) \sim \mathcal{P}_s} \left[\frac{p_t(\mathbf{x})}{p_s(\mathbf{x})} \ell(h(\mathbf{x}), y) \right]$$

The *importance weights* $w(\mathbf{x}) = p_t(\mathbf{x})/p_s(\mathbf{x})$ rebalance the source distribution to match the target. An unbiased estimator of $R_t(h)$ is:

$$\hat{R}_t(h) = \frac{1}{n} \sum_{i=1}^n \hat{w}(\mathbf{x}_i) \ell(h(\mathbf{x}_i), y_i)$$

Density ratio estimation. Estimating $w(\mathbf{x})$ without explicitly estimating p_s and p_t separately is possible via binary classification. If we train a classifier $D : \mathcal{X} \rightarrow [0, 1]$ to distinguish source ($D(\mathbf{x}) \approx 0$) from target ($D(\mathbf{x}) \approx 1$), then:

$$w(\mathbf{x}) = \frac{p_t(\mathbf{x})}{p_s(\mathbf{x})} = \frac{D(\mathbf{x})}{1 - D(\mathbf{x})}$$

Methods include KLIEP (minimise KL divergence), uLSIF (least-squares importance fitting), and deep discriminators (train a neural network to distinguish source from target).

When covariate shift correction fails. Importance weighting works well when p_s and p_t have substantial overlap — specifically, when $\chi^2(p_t \| p_s) = \mathbb{E}_{p_s}[w^2] - 1 < \infty$. If the target distribution has regions with zero source density ($p_s(\mathbf{x}) = 0$ for some $\mathbf{x}$ in the support of p_t), the weights are infinite and no correction is possible. This motivates active learning and domain adaptation approaches that directly collect data from the target distribution.

EXAMPLE — Covariate Shift in Medical Imaging

An X-ray pathology classifier trained at Hospital A (source) is deployed at Hospital B (target). Covariate shift arises from:

- Different scanner hardware (GE vs. Siemens): intensity distributions differ.
- Different patient demographics: different prevalence of incidental findings.
- Different positioning protocols: systematic view-angle differences.

Without correction, the classifier's sensitivity drops from 92% to 74%. With importance-weighted fine-tuning on 500 target examples: sensitivity recovers to 89%. With full fine-tuning on 5,000 target examples: 91.5%.

CORAL: Correlation Alignment. Sun & Saenko (2016) propose aligning the second-order statistics (covariance matrices) of source and target feature distributions. Given source features $\mathbf{F}_S \in \mathbb{R}^{n_S \times d}$ and target features $\mathbf{F}_T \in \mathbb{R}^{n_T \times d}$:

$$\mathcal{L}_{\text{CORAL}} = \frac{1}{4d^2} \|\mathbf{C}_S - \mathbf{C}_T\|_F^2$$

where $\mathbf{C}_S = \frac{1}{n_S-1}(\mathbf{F}_S^\top \mathbf{F}_S - \frac{1}{n_S} \mathbf{1}^\top \mathbf{F}_S^\top \mathbf{F}_S \mathbf{1})$ is the source covariance. CORAL adds this as a regulariser to the source classification loss. For Gaussian distributions, minimising CORAL is equivalent to minimising the Bures metric $d_B(P_S, P_T) = \|\mathbf{C}_S^{1/2} - \mathbf{C}_T^{1/2}\|_F^2$.

Maximum Mean Discrepancy (MMD). An alternative divergence measure for domain adaptation is the MMD, defined using a kernel $k : \mathcal{X} \times \mathcal{X} \rightarrow \mathbb{R}$:

$$\text{MMD}^2(P_S, P_T) = \|\mu_{P_S} - \mu_{P_T}\|_{\mathcal{H}_k}^2 = \mathbb{E}_{P_S}[k(\mathbf{x}, \mathbf{x}')] - 2\mathbb{E}_{P_S, P_T}[k(\mathbf{x}, \mathbf{y})] + \mathbb{E}_{P_T}[k(\mathbf{y}, \mathbf{y}')]$$

where $\mu_P = \mathbb{E}_P[k(\cdot, \mathbf{x})]$ is the kernel mean embedding. With Gaussian kernel $k(\mathbf{x}, \mathbf{y}) = e^{-\|\mathbf{x}-\mathbf{y}\|^2/2\sigma^2}$, MMD can be computed in $O(n^2)$ and differentiated for gradient-based alignment. DAN (Deep Adaptation Network, Long et al. 2015) minimises task loss + MMD across multiple layers.

Optimal transport alignment. Courty et al. (2017) formulate domain adaptation as finding the optimal transport plan π^* between P_S and P_T that minimises transport cost while preserving label structure:

$$\pi^* = \arg \min_{\pi \in \Pi(P_S, P_T)} \sum_{ij} \pi_{ij} \|\mathbf{x}_i^S - \mathbf{x}_j^T\|^2 + \lambda_l \sum_{ij} \pi_{ij} \mathbf{1}[y_i^S \neq \tilde{y}_j^T]$$

The second term penalises transporting examples across class boundaries. After finding π^* , the transported source distribution $\hat{P}_S = \pi^* \mathbf{X}^T$ can be used for domain-adapted training.

Label shift. A dual scenario to covariate shift is *label shift* (Lipton et al. 2018): $p_s(y) \neq p_t(y)$ but $p_s(\mathbf{x}|y) = p_t(\mathbf{x}|y)$. This is corrected by estimating the marginal label distribution shift $w(y) = p_t(y)/p_s(y)$, which can be done from target unlabelled data using the *black-box shift estimation* (BBSE) algorithm.

Domain Adaptation Setting

Source domain $\mathcal{S}$: data from distribution $P_S(\mathbf{x}, y)$.

Target domain $\mathcal{T}$: data from $P_T(\mathbf{x}, y)$, *unlabelled*.

Covariate shift: $P_S(\mathbf{x}) \neq P_T(\mathbf{x})$ but $P_S(y|\mathbf{x}) = P_T(y|\mathbf{x})$.

Label shift: $P_S(y) \neq P_T(y)$ but $P_S(\mathbf{x}|y) = P_T(\mathbf{x}|y)$.

46.2 The Ben-David Bound

$\mathcal{H}$ -divergence

For a hypothesis class $\mathcal{H}$, the $\mathcal{H}$ -divergence between P_S and P_T is:

$$d_{\mathcal{H}}(P_S, P_T) = 2 \sup_{h \in \mathcal{H}} |P_{P_S}[h(\mathbf{x}) = 1] - P_{P_T}[h(\mathbf{x}) = 1]|$$

It measures how well any classifier in $\mathcal{H}$ can distinguish the two domains.

Ben-David Theorem. For any $h \in \mathcal{H}$:

$$\varepsilon_T(h) \leq \varepsilon_S(h) + \frac{1}{2} d_{\mathcal{H}}(P_S, P_T) + \lambda$$

where $\lambda = \min_{h^* \in \mathcal{H}} \varepsilon_S(h^*) + \varepsilon_T(h^*)$ is the ideal joint error.

Implication. To minimise target error, we need: (1) low source error ($\varepsilon_S(h)$ small), (2) indistinguishable domains ($d_{\mathcal{H}}$ small), (3) a λ small (there exists a good common hypothesis).

46.3 Domain-Adversarial Neural Network (DANN)

DANN (Ganin & Lempitsky, 2015) is the algorithmic realisation of the Ben-David bound: it minimises source risk while simultaneously minimising $d_{\mathcal{H}}$ by learning domain-invariant representations via adversarial training.

Architecture. DANN consists of three components:

- **Feature extractor** $G_f : \mathcal{X} \rightarrow \mathcal{Z}$ (parameters θ_f)
- **Label predictor** $G_y : \mathcal{Z} \rightarrow \mathcal{Y}$ (parameters θ_y), trained on source data
- **Domain discriminator** $G_d : \mathcal{Z} \rightarrow \{0, 1\}$ (parameters θ_d), predicting source vs. target

Minimax objective. The training objective is:

$$\min_{\theta_f, \theta_y} \max_{\theta_d} E(\theta_f, \theta_y, \theta_d)$$

where:

$$E = \frac{1}{n_S} \sum_{i \in S} \mathcal{L}_y(G_y(G_f(\mathbf{x}_i)), y_i) - \lambda \left[\frac{1}{n_S + n_T} \sum_{i \in S \cup T} \mathcal{L}_d(G_d(G_f(\mathbf{x}_i)), d_i) \right]$$

Here $d_i = 0$ for source, $d_i = 1$ for target, and $\lambda > 0$ controls the trade-off.

The gradient reversal layer (GRL). The minimax can be solved as a standard gradient descent by inserting a *gradient reversal layer* between G_f and G_d :

$$\text{GRL}(\mathbf{z}) = \mathbf{z} \quad (\text{forward pass}), \quad \frac{\partial \text{GRL}(\mathbf{z})}{\partial \mathbf{z}} = -\lambda \mathbf{I} \quad (\text{backward pass})$$

During the forward pass, the GRL acts as an identity. During backpropagation, it negates and scales the gradient. Thus:

- G_d receives standard gradients $\rightarrow$ it improves at distinguishing source from target.
- G_f receives negated gradients from $G_d \rightarrow$ it tries to *confuse* G_d , making source and target features indistinguishable.
- G_y receives standard gradients from the label loss $\rightarrow$ it learns to classify correctly on source data.

EXAMPLE — DANN Training Step

Mini-batch: 32 source samples (x_i^S, y_i) and 32 unlabelled target samples x_j^T .

Forward: $\mathbf{z}_i^S = G_f(x_i^S)$, $\mathbf{z}_j^T = G_f(x_j^T)$.

Label loss: $\mathcal{L}_y = -\frac{1}{32} \sum_i \log G_y(\mathbf{z}_i^S)_{y_i}$.

Domain loss: $\mathcal{L}_d = -\frac{1}{64} \left[\sum_i \log G_d(\mathbf{z}_i^S) + \sum_j \log(1 - G_d(\mathbf{z}_j^T)) \right]$.

Backward: gradients of $\mathcal{L}_y$ flow normally; gradients of $\mathcal{L}_d$ through GRL are negated before reaching G_f .

Result: G_f learns features that are predictive of labels but indistinguishable as source or target.

Connection to Ben-David bound. When the domain discriminator is trained to optimality, $\mathcal{L}_d = H(\mathcal{H}, \mathcal{D}_S, \mathcal{D}_T)$ (the $\mathcal{H}$ -divergence). Minimising $\mathcal{L}_d$ in the feature space directly

minimises $d_{\mathcal{H}}$ in the Ben-David bound, providing a principled justification for the DANN objective.

DANN minimises source classification loss while maximising domain confusion (minimising $d_{\mathcal{H}}$):

$$\min_{\square_f, \square_y} \max_{\square_d} \frac{1}{n_s} \sum_{i=1}^{n_s} \mathcal{L}_y(G_y(G_f(\mathbf{x}_i^s)), y_i) - \lambda \cdot \frac{1}{n_s + n_t} \sum_i \mathcal{L}_d(G_d(G_f(\mathbf{x}_i)), d_i)$$

where G_f is the feature extractor, G_y the classifier, G_d the domain discriminator, and $d_i \in \{0, 1\}$ the domain label.

Gradient Reversal Layer (GRL). During the forward pass the GRL acts as the identity; during backpropagation it multiplies the gradient by $-\lambda$, effectively performing gradient *ascent* on the domain loss with respect to $\square_f$. This allows standard SGD to optimise the minimax objective in a single pass.

EXAMPLE — GRL gradient flip

Forward: $\mathbf{h} = G_f(\mathbf{x})$ passes unchanged through GRL.

Backward: gradient arriving at GRL is $\frac{\partial \mathcal{L}_d}{\partial \mathbf{h}} = \mathbf{g}$. GRL outputs $-\lambda \mathbf{g}$ to G_f .

Net effect: G_f updates so as to *increase* domain loss — making features domain-invariant.

46.4 Self-Training and Pseudo-Labels for Domain Adaptation

When no labelled target data is available and the Ben-David bound's $d_{\mathcal{H}}$ term cannot be directly minimised, self-training provides an alternative: iteratively generate pseudo-labels for target data and train on them.

Self-training algorithm.

1. Train a model f_θ on labelled source data $\mathcal{D}_S$.
2. Predict pseudo-labels on unlabelled target data: $\tilde{y}_j = \arg \max_c f_\theta(x_j^T)$.
3. Filter: keep only pseudo-labels with confidence $\max_c f_\theta(x_j^T)_c \geq \tau$ (threshold $\tau \in [0.9, 0.95]$ typically).
4. Retrain f_θ on $\mathcal{D}_S \cup \{(x_j^T, \tilde{y}_j) : \text{high-confidence}\}$.
5. Repeat until convergence.

Why self-training works. The Cluster Assumption: the decision boundary should pass through low-density regions. High-confidence predictions are correct precisely when the input lies near the cluster centres. Pseudo-labelling reinforces the cluster structure by treating high-confidence predictions as true, which enlarges the margin on target data.

Theoretical analysis. Under the cluster assumption, self-training reduces the Ben-David λ^* term: the “joint error” of the best hypothesis on both domains decreases as the model learns target-domain structure via pseudo-labels.

EXAMPLE — Self-Training on VisDA-C (Synthetic → Real)

Source: 3D synthetic renderings of 12 object categories ($n_S = 280K$). Target: MS-COCO natural images ($n_T = 55K$, unlabelled at training time).

Results:

Method	Target Accuracy
Source only (no adaptation)	45.2%
DANN	57.4%
Self-training ($\tau = 0.9$, 3 rounds)	68.3%
DANN + Self-training	74.1%

Self-training alone outperforms DANN by 11% and is complementary to it: combining both achieves 74%, as DANN aligns distributions while self-training refines the target decision boundary.

Connection to semi-supervised learning. Self-training with pseudo-labels is equivalent to EM with a hard assignment step: the E-step assigns labels (pseudo-labels), the M-step maximises the likelihood (retraining). Consistency regularisation methods (Mean Teacher, FixMatch) are soft versions that use the model’s own predictions as soft targets rather than hard labels.

46.5 FixMatch and Consistency Regularisation

FixMatch (Sohn et al., 2020) combines pseudo-labelling and consistency regularisation into a simple yet state-of-the-art semi-supervised learning algorithm, achieving near-supervised performance with very few labelled examples.

Consistency regularisation principle. A natural prior for semi-supervised learning is the *cluster assumption*: the decision boundary should pass through low-density regions, and hence nearby points (related by a data augmentation) should have the same label. Formally, for unlabelled example $\mathbf{x}$ and augmentation $\mathcal{A}$:

$$\min_{\theta} D(f_{\theta}(\mathbf{x}) \| f_{\theta}(\mathcal{A}(\mathbf{x})))$$

where D is a divergence (KL, mean squared error). This forces the model to produce consistent predictions under perturbations of the input.

FixMatch algorithm. For each unlabelled example $\mathbf{x}^u$:

1. Compute “soft” pseudo-label using weak augmentation: $\hat{y} = \arg \max_c f_{\theta}(\mathcal{A}_w(\mathbf{x}^u))$.
2. **Threshold:** use the pseudo-label only if $\max_c f_{\theta}(\mathcal{A}_w(\mathbf{x}^u)) \geq \tau$ (typically $\tau = 0.95$).
3. Compute strongly-augmented input $\mathcal{A}_s(\mathbf{x}^u)$ and enforce consistency with the pseudo-label:

$$\mathcal{L}_u = \mathbf{1}[\max_c p_c \geq \tau] \cdot H(\hat{y}, f_{\theta}(\mathcal{A}_s(\mathbf{x}^u)))$$

The total FixMatch loss: $\mathcal{L} = \mathcal{L}_s + \lambda_u \mathcal{L}_u$.

Convergence analysis. Under the Lipschitz assumption on f_{θ} and the cluster assumption (high-confidence predictions near true clusters), FixMatch converges to the supervised optimal:

$$R_{\text{FixMatch}} \leq R_{\text{supervised}} + O\left(\frac{\text{error in pseudo-labels}}{\sqrt{n_u}}\right)$$

where n_u is the number of unlabelled examples. The error in pseudo-labels is controlled by the threshold τ : higher τ means fewer pseudo-labels but higher quality.

EXAMPLE — FixMatch Performance on CIFAR-10

Method	Labels	Test Error
Supervised	250	36.2%
Pseudo-label	250	28.2%
MixMatch	250	11.1%
FixMatch	250	7.5%
Supervised (full)	50000	4.1%

With only 250 labelled examples (0.5% of the training set), FixMatch achieves 7.5% error — within 3.4% of the fully-supervised baseline that uses 200× more labels. The combination of strong augmentation (RandAugment + Cutout) and thresholded pseudo-labels is key to the strong performance.

Connection to domain adaptation. FixMatch can be viewed as a special case of DANN (§46.3) where the “domain” is defined by the augmentation type: weak-augmented images are the “source domain” and strong-augmented images are the “target domain”. The consistency regularisation minimises the domain divergence $d_{\mathcal{H}}(P_{\text{weak}}, P_{\text{strong}})$ in the feature space, exactly as the Ben-David bound requires.

Mean Teacher. Tarvainen & Valpola (2017) propose a smooth version of self-training: maintain an *exponential moving average* (EMA) teacher model $\bar{\theta}$ alongside the student model θ :

$$\bar{\theta}_t = (1 - \alpha)\bar{\theta}_{t-1} + \alpha\theta_t$$

The teacher generates pseudo-labels from its more stable predictions; the student is trained to match the teacher. The EMA teacher is strictly more accurate than the student at any point in training (since it averages over many student checkpoints), providing higher-quality targets. Mean Teacher achieves consistent improvements over fixed pseudo-labelling across semi-supervised benchmarks.

46.6 Structured Covariate Shift and Joint Distribution Alignment

Standard domain adaptation methods align the marginal distributions $P_S(\mathbf{x})$ and $P_T(\mathbf{x})$. When the class-conditional distributions also differ ($P_S(y|\mathbf{x}) \neq P_T(y|\mathbf{x})$), *joint distribution alignment* is required.

Joint distribution discrepancy. Define the joint distribution distance:

$$d_J(P_S, P_T) = d(P_S(\mathbf{x}, y), P_T(\mathbf{x}, y))$$

For the maximum mean discrepancy:

$$\text{JMD}(P_S, P_T) = \sum_{c=1}^C \|\mu_{P_S^c} - \mu_{P_T^c}\|_{\mathcal{H}_k}^2$$

where $P_S^c = P_S(\mathbf{x}|y = c)$ is the class-conditional source distribution. JMD requires pseudo-labels $\tilde{y}$ for target examples to compute $\mu_{P_T^c}$.

Joint Distribution Adaptation (JDA). Long et al. (2013) minimise both marginal and conditional MMD:

$$\min_{\theta} \hat{R}_S(h_{\theta}) + \lambda \left(\text{MMD}^2(P_S(\mathbf{z}), P_T(\mathbf{z})) + \sum_c \text{MMD}^2(P_S^c(\mathbf{z}), \tilde{P}_T^c(\mathbf{z})) \right)$$

where $\mathbf{z} = g_\theta(\mathbf{x})$ are the learned features. The target pseudo-labels $\tilde{y}$ are initialised by a source-trained classifier and updated alternately with the features.

Importance weighting for structured shift. Under label shift (§46.3), the density ratio is:

$$w(\mathbf{x}) = \frac{p_T(\mathbf{x})}{p_S(\mathbf{x})} = \frac{p_T(y)}{p_S(y)} \quad (\text{under label shift assumption})$$

The label ratio $p_T(y)/p_S(y)$ can be estimated without target labels by solving:

$$p_S(f(\mathbf{x})) = \sum_c p_S(f(\mathbf{x})|y=c)p_S(y=c)$$

$$p_T(f(\mathbf{x})) = \sum_c p_T(f(\mathbf{x})|y=c)p_T(y=c)$$

Under the label shift assumption, $p_S(f(\mathbf{x})|y=c) = p_T(f(\mathbf{x})|y=c)$, giving the BBSE estimator:

$$\hat{p}_T(y) = \mathbf{C}_S^{-1} \hat{p}_T(f(\mathbf{x}))$$

where $\mathbf{C}_S \in \mathbb{R}^{C \times C}$ is the confusion matrix of the source classifier. This allows accurate label shift correction without target labels.

EXAMPLE — Covariate Shift Correction on Office-31

Office-31 dataset: Amazon $\rightarrow$ DSLR adaptation (31 object classes, $n_S = 2817$, $n_T = 498$).

Methods:

Method	Target Accuracy
No adaptation (source only)	63.8%
CORAL (2nd-order alignment)	75.4%
DAN (MMD, 3 layers)	78.6%
DANN (domain adversarial)	79.7%
JDA (joint distribution)	80.9%
CDAN (conditional DANN)	82.7%

Aligning both marginal and conditional distributions (JDA, CDAN) consistently outperforms marginal-only methods, by $\sim 2\text{--}3\%$. For tasks with large class-conditional shift (different background textures per domain), joint alignment is essential.

Adversarial discriminative alignment. Tzeng et al. (2017) propose ADDA (Adversarial Discriminative Domain Adaptation), which trains source and target encoders separately with an adversarial discriminator:

1. Pre-train source encoder f_S and classifier h on source data.
2. Fix f_S ; adversarially train target encoder f_T so its features are indistinguishable from f_S 's features (discriminator tries to distinguish domains; f_T tries to fool it).
3. At test time, use $h \circ f_T$ for target prediction.

ADDA allows the target encoder to be more flexible than DANN (which uses a shared encoder), at the cost of not benefiting from target feature feedback during source training.

Neural ODEs and Continuous-Depth Networks

Residual networks become neural ODEs in the continuous limit. We derive the adjoint method for memory-efficient gradient computation through an ODE solver, and discuss the connection to normalising flows.

47.1 From ResNets to ODEs

A ResNet update is $\mathbf{h}_{t+1} = \mathbf{h}_t + f_{\square}(\mathbf{h}_t, t)$. In the continuous limit (step size $\rightarrow 0$):

$$\begin{aligned}\frac{d\mathbf{h}(t)}{dt} &= f_{\square}(\mathbf{h}(t), t), & \mathbf{h}(0) &= \mathbf{x} \\ \mathbf{h}(T) &= \mathbf{h}(0) + \int_0^T f_{\square}(\mathbf{h}(t), t) dt\end{aligned}$$

The “depth” is now a continuous variable $t \in [0, T]$. The output $\mathbf{h}(T)$ is fed to a final layer for prediction.

47.2 The Adjoint Method

The adjoint method is the continuous analogue of backpropagation through time. It computes gradients of an ODE-based loss without storing intermediate states, reducing memory from $O(T)$ to $O(1)$ in the number of integration steps.

Problem setup. Given the ODE:

$$\frac{d\mathbf{h}(t)}{dt} = f_{\theta}(\mathbf{h}(t), t), \quad \mathbf{h}(t_0) = \mathbf{h}_0$$

and loss $\mathcal{L} = L(\mathbf{h}(t_1))$ at the final time t_1 , we want $\partial\mathcal{L}/\partial\theta$ and $\partial\mathcal{L}/\partial\mathbf{h}_0$.

Adjoint state. Define the adjoint $\mathbf{a}(t) = \partial\mathcal{L}/\partial\mathbf{h}(t)$. At t_1 : $\mathbf{a}(t_1) = \partial L/\partial\mathbf{h}(t_1)$ (backpropagation through the final loss).

Adjoint ODE (backward-in-time). The adjoint satisfies:

$$\frac{d\mathbf{a}(t)}{dt} = -\mathbf{a}(t)^{\top} \frac{\partial f_{\theta}}{\partial \mathbf{h}}$$

Derivation. Differentiate the ODE constraint with respect to $\mathbf{h}(t_0)$, obtaining the variational equation for the sensitivity $\partial\mathbf{h}(t)/\partial\mathbf{h}(t_0)$. Then apply the chain rule:

$$\frac{d}{dt}\mathbf{a}(t) = \frac{d}{dt} \frac{\partial\mathcal{L}}{\partial\mathbf{h}(t)} = \frac{\partial\mathcal{L}}{\partial\mathbf{h}(t)} \cdot \frac{\partial}{\partial\mathbf{h}(t)} \left(\frac{d\mathbf{h}}{dt} \right) \cdot \left(\frac{\partial\mathbf{h}(t+dt)}{\partial\mathbf{h}(t)} \right)^{-1}$$

Using the first-order expansion $\mathbf{h}(t+dt) \approx \mathbf{h}(t) + f_{\theta}dt$, so $\partial\mathbf{h}(t+dt)/\partial\mathbf{h}(t) \approx \mathbf{I} + (\partial f_{\theta}/\partial\mathbf{h})dt$:

$$\frac{d\mathbf{a}(t)}{dt} = -\mathbf{a}(t)^{\top} \frac{\partial f_{\theta}(\mathbf{h}(t), t)}{\partial \mathbf{h}(t)}$$

Parameter gradient. The gradient with respect to θ is:

$$\frac{d\mathcal{L}}{d\theta} = - \int_{t_1}^{t_0} \mathbf{a}(t)^{\top} \frac{\partial f_{\theta}(\mathbf{h}(t), t)}{\partial \theta} dt$$

Derivation. By the chain rule and the ODE:

$$\frac{d\mathcal{L}}{d\theta} = \int_{t_0}^{t_1} \frac{\partial \mathcal{L}}{\partial \mathbf{h}(t)} \cdot \frac{\partial \dot{\mathbf{h}}(t)}{\partial \theta} dt = \int_{t_0}^{t_1} \mathbf{a}(t)^\top \frac{\partial f_\theta}{\partial \theta} dt$$

The sign difference arises from the backward-in-time integration.

Adjoint Method Algorithm

1. **Forward pass:** integrate $\dot{\mathbf{h}} = f_\theta(\mathbf{h}, t)$ from t_0 to t_1 . Store only $\mathbf{h}(t_1)$.
2. **Initialise adjoint:** $\mathbf{a}(t_1) = \partial L / \partial \mathbf{h}(t_1)$.
3. **Backward pass:** integrate backward from t_1 to t_0 the augmented ODE:

$$\frac{d}{dt} \begin{pmatrix} \mathbf{h} \\ \mathbf{a} \\ \mathbf{g}_\theta \end{pmatrix} = \begin{pmatrix} f_\theta(\mathbf{h}, t) \\ -\mathbf{a}^\top \partial f_\theta / \partial \mathbf{h} \\ -\mathbf{a}^\top \partial f_\theta / \partial \theta \end{pmatrix}$$

starting from $(\mathbf{h}(t_1), \mathbf{a}(t_1), \mathbf{0})$.

4. **Output:** $\mathbf{a}(t_0) = \partial \mathcal{L} / \partial \mathbf{h}_0$, $\mathbf{g}_\theta(t_0) = \partial \mathcal{L} / \partial \theta$.

Memory: $O(1)$ — no stored intermediate states (the backward ODE solver recomputes $\mathbf{h}(t)$ on the fly).

EXAMPLE — Memory comparison: Backprop vs. Adjoint for 1000 ODE steps

Backprop through the solver: store all $\mathbf{h}(t_1), \dots, \mathbf{h}(t_{1000})$. For $d = 512$ hidden dims, fp32: $1000 \times 512 \times 4\text{B} = 2.05\text{ MB}$ per trajectory.

Adjoint method: store only $\mathbf{h}(t_1) \in \mathbb{R}^{512}$: 2.05 KB. A $1000\times$ memory reduction.

For sequence length 100K and batch size 32: backprop requires $\sim 6.4\text{ GB}$, adjoint $\sim 6.4\text{ MB}$.

Trade-off: the adjoint backward pass requires a second call to the ODE solver, approximately doubling wall-clock time. The memory saving is thus exchanged for time.

To train, we need $\frac{\partial \mathcal{L}}{\partial \square}$. Backpropagating through the ODE solver's intermediate states requires $O(T)$ memory. The adjoint method avoids storing the forward trajectory.

Define the adjoint state $\mathbf{a}(t) = \frac{\partial \mathcal{L}}{\partial \mathbf{h}(t)}$. It satisfies the adjoint ODE:

$$\frac{d\mathbf{a}(t)}{dt} = -\mathbf{a}(t)^\top \frac{\partial f_\square}{\partial \mathbf{h}(t, t)}$$

Gradient computation. Augment the state with the parameter gradient:

$$\frac{d}{dt} \frac{\partial \mathcal{L}}{\partial \square} = -\mathbf{a}(t)^\top \frac{\partial f_\square}{\partial \square}$$

The full procedure: (1) solve the forward ODE from $t = 0$ to T ; (2) solve the augmented reverse ODE from T to 0, obtaining $\mathbf{a}(0)$ and $\frac{\partial \mathcal{L}}{\partial \square}$ simultaneously. Memory cost: $O(1)$ (only the current state is needed).

EXAMPLE — Adjoint vs backpropagation on $T = 100$ steps

Standard BPTT through RK4 ODE solver: stores 400 function evaluations $\times$ hidden dimension. For $d = 512$: 200K floats $\approx$ 800KB.

Adjoint method: stores current $\mathbf{h}(t)$, $\mathbf{a}(t)$, and $\nabla_{\square} \mathcal{L}$ — three vectors of size 512: 6KB. $\approx 130\times$ memory reduction.

47.3 Continuous Normalising Flows

Continuous Normalising Flows (CNFs) (Chen et al., 2018) extend discrete normalising flows (§18 in Volume II) by defining the flow as the solution of an ODE, enabling flexible density models with $O(1)$ memory training via the adjoint method.

Setup. A normalising flow maps a simple base distribution $p_0(\mathbf{z})$ (e.g., $\mathcal{N}(0, \mathbf{I})$) to a complex target distribution $p_1(\mathbf{x})$ via an invertible transformation $\mathbf{x} = T(\mathbf{z})$. The density is:

$$\log p_1(\mathbf{x}) = \log p_0(T^{-1}(\mathbf{x})) - \log \left| \det \frac{\partial T}{\partial \mathbf{z}} \right|$$

In a CNF, T is defined implicitly as the flow of an ODE:

$$\frac{d\mathbf{z}(t)}{dt} = f_{\theta}(\mathbf{z}(t), t), \quad \mathbf{z}(0) \sim p_0$$

The mapping $\mathbf{z}(0) \mapsto \mathbf{z}(1)$ is T , and $\mathbf{z}(1) = \mathbf{x}$ is the generated sample.

Instantaneous change of variables. The log-density evolves as:

$$\frac{d \log p(\mathbf{z}(t), t)}{dt} = -\text{tr} \left(\frac{\partial f_{\theta}}{\partial \mathbf{z}} \right)$$

Proof. The Jacobian of the flow $\partial \mathbf{z}(t) / \partial \mathbf{z}(0)$ satisfies:

$$\frac{d}{dt} \ln \left| \det \frac{\partial \mathbf{z}(t)}{\partial \mathbf{z}(0)} \right| = \text{tr} \left(\frac{\partial f_{\theta}}{\partial \mathbf{z}} \right)$$

by Jacobi's formula $d \ln |\det \mathbf{A}| / dt = \text{tr}(\mathbf{A}^{-1} \dot{\mathbf{A}})$ combined with the ODE for the Jacobian. The log-density then changes at the negative rate.

Log-likelihood computation.

$$\log p_1(\mathbf{x}) = \log p_0(\mathbf{z}(0)) - \int_0^1 \text{tr} \left(\frac{\partial f_{\theta}}{\partial \mathbf{z}} \right) dt$$

Computing the trace $\text{tr}(\partial f_{\theta} / \partial \mathbf{z})$ naively requires d backward passes (one per output dimension). Instead, use the Hutchinson estimator:

$$\text{tr} \left(\frac{\partial f_{\theta}}{\partial \mathbf{z}} \right) \approx \square^{\top} \frac{\partial f_{\theta}}{\partial \mathbf{z}} \square, \quad \square \sim \mathcal{N}(0, \mathbf{I})$$

This reduces trace computation from $O(d)$ backward passes to $O(1)$ (with higher variance).

EXAMPLE — CNF vs. Discrete Flow: Parameter Comparison for 2D Density

Target: a 2D distribution with complex multi-modal structure ($K = 8$ modes).

Discrete RealNVP (10 coupling layers, 64 hidden units): $\sim 40,000$ parameters.

CNF (ODE with 3-layer MLP, 64 hidden units): $\sim 12,000$ parameters.
 The CNF achieves comparable density estimation quality with $3\times$ fewer parameters.
 The key: discrete flows must discretise time, requiring $O(T)$ coupling layers for T time steps. The CNF uses a single continuous trajectory, naturally handling any resolution.

Stability analysis of neural ODEs. For the ODE $\dot{\mathbf{h}} = f_\theta(\mathbf{h})$ (autonomous, time-invariant), stability near an equilibrium $\mathbf{h}^* = 0$ is determined by the Jacobian $\mathbf{J} = \nabla_{\mathbf{h}} f_\theta(\mathbf{h}^*)$:

- *Asymptotically stable*: all eigenvalues of $\mathbf{J}$ have negative real parts ($\text{Re}(\lambda_i) < 0$). The solution converges to $\mathbf{h}^*$.
- *Marginally stable*: all eigenvalues have $\text{Re}(\lambda_i) \leq 0$. No growth or decay.
- *Unstable*: some eigenvalue has $\text{Re}(\lambda_i) > 0$. The solution diverges.

For classification, the neural ODE should drive representations toward a stable equilibrium near class centroids. Anti-symmetric parametrisations $f_\theta(\mathbf{h}) = \mathbf{W}\mathbf{h}$ with $\mathbf{W}$ antisymmetric ($\mathbf{W}^\top = -\mathbf{W}$) guarantee stability: eigenvalues of antisymmetric matrices are purely imaginary, giving stable oscillations (Lie bracket constraint).

The forward-backward symmetry. Unlike discrete ResNets, neural ODEs are reversible: running the ODE backward in time recovers the input exactly (up to numerical error). This reversibility is exploited for:

- *Memory-efficient training*: never store activations; recompute them during the backward pass by integrating the ODE backward. This is the adjoint method (§47.2).
- *Invertible generative models*: the CNF defines a bijection between $\mathcal{N}(0, \mathbf{I})$ and the data distribution, enabling exact likelihood computation (unlike VAEs) and exact inversion (unlike diffusion models).
- *Normalisation flows*: FFJORD extends RealNVP to free-form (non-triangular) vector fields at the cost of ODE integration, enabling more expressive flows with fewer parameters.

EXAMPLE — ODE Integration in Practice: Dormand-Prince RK45

Standard adaptive ODE solver for neural ODEs: Runge-Kutta-4-5 (RK45) or Dormand-Prince.

Per-step error estimate: $e_{t+1} = |y_{\text{RK4}}(t+h) - y_{\text{RK5}}(t+h)|$. Adaptive step size: $h_{t+1} = h_t \cdot \min(h_{\text{max}}/h_t, S \cdot (e_{\text{tol}}/e_{t+1})^{0.2})$ with safety factor $S = 0.9$.

For a neural ODE with $d = 512$, tolerance $e_{\text{tol}} = 10^{-3}$: typically 4–10 function evaluations (NFE) for $t \in [0, 1]$ with smooth dynamics. With stiff dynamics (very large Jacobian eigenvalues): may need implicit solvers (e.g., Crank-Nicolson, SDIRK) with 10^3 – 10^4 NFE.

Training cost: NFE per step $\times$ cost of one forward pass. The adjoint doubles NFE but removes the $O(L)$ activation storage of a corresponding discrete ResNet with $L = \text{NFE}$ layers.

FFJORD: efficient CNF training. Grathwohl et al. (2018) combine the adjoint method (§47.2) with the Hutchinson trace estimator, giving Free-Form Jacobian of Reversible Dynamics (FFJORD). Training cost: $O(P)$ per time step (same as a forward pass), with $O(1)$ memory via the adjoint. This makes CNFs scalable to image-level densities.

Neural ODEs define invertible maps (since the ODE can be solved forward and backward). The change-of-variables formula for the log-density is:

$$\log p(\mathbf{h}(T)) = \log p(\mathbf{h}(0)) - \int_0^T \text{tr} \left(\frac{\partial f_{\square}}{\partial \mathbf{h}(t)} \right) dt$$

Computing the full Jacobian trace is $O(d^2)$, but the Hutchinson estimator gives an unbiased stochastic estimate in $O(d)$:

$$\text{tr} \left(\frac{\partial f}{\partial \mathbf{h}} \right) = \mathbb{E}_{\square \sim \mathcal{N}(\mathbf{0}, \mathbf{I})} \left[\square^\top \frac{\partial f}{\partial \mathbf{h}} \square \right]$$

47.4 Physics-Informed Neural Networks (PINNs)

Physics-Informed Neural Networks (Raissi, Perdikaris & Karniadakis 2019) use the ODE/PDE framework to incorporate physical constraints directly into the neural network loss, enabling accurate solutions to differential equations with minimal labelled data.

Setup. Consider a general PDE with solution $u(\mathbf{x}, t)$:

$$\mathcal{N}[u](\mathbf{x}, t) = f(\mathbf{x}, t), \quad (\mathbf{x}, t) \in \Omega \times [0, T]$$

with boundary conditions $u(\mathbf{x}, t) = g(\mathbf{x}, t)$ on $\partial\Omega$ and initial condition $u(\mathbf{x}, 0) = u_0(\mathbf{x})$.

A PINN approximates u with a neural network $u_\theta(\mathbf{x}, t)$ and minimises:

$$\mathcal{L}_{\text{PINN}}(\theta) = w_r \mathcal{L}_r + w_b \mathcal{L}_b + w_0 \mathcal{L}_0$$

where:

- **Residual loss:** $\mathcal{L}_r = \frac{1}{N_r} \sum_{i=1}^{N_r} |\mathcal{N}[u_\theta](\mathbf{x}_i^r, t_i^r) - f(\mathbf{x}_i^r, t_i^r)|^2$ (PDE satisfied at collocation points).
- **Boundary loss:** $\mathcal{L}_b = \frac{1}{N_b} \sum_{i=1}^{N_b} |u_\theta(\mathbf{x}_i^b, t_i^b) - g(\mathbf{x}_i^b, t_i^b)|^2$.
- **Initial loss:** $\mathcal{L}_0 = \frac{1}{N_0} \sum_{i=1}^{N_0} |u_\theta(\mathbf{x}_i^0, 0) - u_0(\mathbf{x}_i^0)|^2$.

Automatic differentiation for PDE residuals. The residual $\mathcal{N}[u_\theta]$ involves derivatives of u_θ with respect to $\mathbf{x}$ and t , computed via automatic differentiation. For the heat equation $\partial_t u = \kappa \Delta u$:

$$\mathcal{L}_r = \frac{1}{N_r} \sum_i \left(\frac{\partial u_\theta}{\partial t}(\mathbf{x}_i, t_i) - \kappa \sum_{j=1}^d \frac{\partial^2 u_\theta}{\partial x_j^2}(\mathbf{x}_i, t_i) \right)^2$$

Each evaluation of $\partial^2 u_\theta / \partial x_j^2$ costs $O(d)$ backward passes, making the full residual loss $O(d^2)$ per collocation point (for d -dimensional PDEs). Efficient implementations batch the Hessian computation.

Convergence theory for PINNs. Mishra & Molinaro (2022) prove that under regularity conditions on u and using N_r collocation points, the PINN error satisfies:

$$\|u - u_\theta\|_{L^2(\Omega)} \leq C \left(\text{approx error} + \sqrt{\frac{\mathcal{L}_r^{\min}}{N_r}} + \frac{1}{\sqrt{N_r}} \right)$$

The error decreases as $O(1/\sqrt{N_r})$ in the number of collocation points, paralleling MC estimation. The approximation error depends on network width/depth.

EXAMPLE — PINN for the 1D Burgers Equation

Burgers equation: $\partial_t u + u \partial_x u = \nu \partial_{xx} u$, $(x, t) \in [-1, 1] \times [0, 1]$, $\nu = 0.01/\pi$.

PINN architecture: 8-layer MLP, tanh activations, 20 neurons per layer (≈ 2000 parameters).

Collocation points: $N_r = 10000$ interior, $N_b = 200$ boundary, $N_0 = 100$ initial.

Training: L-BFGS (full-batch) for 50000 iterations.

Result: L^2 relative error 3×10^{-3} vs. exact solution — comparable to a fine-grid finite difference solver with 128^2 grid points, using $64\times$ fewer degrees of freedom. The PINN achieves mesh-free, continuous-space solution representation.

Failure modes and remedies. PINNs are effective for smooth PDEs but struggle with:

- *Sharp gradients/shocks*: the tanh network is biased toward smooth solutions. Remedy: adaptive collocation (more points near shocks), Fourier features for oscillatory solutions.
- *Stiff equations*: large time-scale ratios cause gradient imbalance. Remedy: temporal decomposition (solve on sub-intervals), causal training (enforce temporal causality).
- *High-dimensional PDEs*: $O(d^2)$ cost of Hessian computation. Remedy: random feature methods, Hutchinson trace estimator (§47.3).

47.5 Diffusion Models as Stochastic Differential Equations

Score-based generative models (Song et al., 2020) unify DDPM, NCSN, and flow-based models under the SDE framework, providing a mathematically coherent foundation and enabling continuous-time sampling.

The forward SDE. Data is progressively corrupted by a forward SDE:

$$d\mathbf{x} = f(\mathbf{x}, t) dt + g(t) d\mathbf{W}$$

Common choices:

- *VPSDE (DDPM)*: $f(\mathbf{x}, t) = -\frac{1}{2}\beta(t)\mathbf{x}$, $g(t) = \sqrt{\beta(t)}$. The marginal $p_t(\mathbf{x})$ converges to $\mathcal{N}(0, \mathbf{I})$ as $t \rightarrow T$.
- *VESDE (NCSN)*: $f = 0$, $g(t) = \sqrt{d(\sigma^2(t))/dt}$. The variance $\sigma^2(t)$ grows; no drift.

The reverse SDE. By Anderson (1982), the time-reversal of an Itô SDE is:

$$d\mathbf{x} = [f(\mathbf{x}, t) - g(t)^2 \nabla_{\mathbf{x}} \log p_t(\mathbf{x})] dt + g(t) d\bar{\mathbf{W}}$$

where $\bar{\mathbf{W}}$ is a reverse-time Brownian motion. The term $\nabla_{\mathbf{x}} \log p_t(\mathbf{x})$ is the *score function* of the marginal density at time t . The reverse SDE generates samples by starting from $p_T \approx \mathcal{N}(0, \mathbf{I})$ and running the time-reversed dynamics.

Score matching. The score function $s_\theta(\mathbf{x}, t) \approx \nabla_{\mathbf{x}} \log p_t(\mathbf{x})$ is learned by minimising:

$$\mathcal{L}_{\text{SM}} = \mathbb{E}_{t, \mathbf{x}_0, \mathbf{x}_t} [\lambda(t) \|s_\theta(\mathbf{x}_t, t) - \nabla_{\mathbf{x}_t} \log p_{0t}(\mathbf{x}_t | \mathbf{x}_0)\|^2]$$

For the VPSDE, $p_{0t}(\mathbf{x}_t | \mathbf{x}_0) = \mathcal{N}(\alpha(t)\mathbf{x}_0, \sigma^2(t)\mathbf{I})$, so:

$$\nabla_{\mathbf{x}_t} \log p_{0t}(\mathbf{x}_t | \mathbf{x}_0) = -\frac{\mathbf{x}_t - \alpha(t)\mathbf{x}_0}{\sigma^2(t)} = -\frac{\square}{\sigma(t)}$$

where $\mathbf{x}_t = \alpha(t)\mathbf{x}_0 + \sigma(t)\boldsymbol{\epsilon}$, $\boldsymbol{\epsilon} \sim \mathcal{N}(0, \mathbf{I})$. The score matching objective becomes ϵ -prediction (the DDPM objective):

$$\mathcal{L}_{\text{DDPM}} = \mathbb{E}_{t, \mathbf{x}_0, \boldsymbol{\epsilon}} [\|\boldsymbol{\epsilon} - \boldsymbol{\epsilon}_\theta(\mathbf{x}_t, t)\|^2]$$

Probability flow ODE. Every SDE has an associated ODE with the same marginals:

$$d\mathbf{x} = \left[f(\mathbf{x}, t) - \frac{1}{2}g(t)^2 \nabla_{\mathbf{x}} \log p_t(\mathbf{x}) \right] dt$$

This is the *probability flow ODE*. Unlike the SDE (stochastic), this is deterministic. It enables:

- Exact likelihood computation: $\log p_0(\mathbf{x}_0) = \log p_T(\mathbf{x}_T) + \int_0^T \nabla \cdot f_{\text{ODE}}(\mathbf{x}(t), t) dt$ (instantaneous change of variables formula for ODEs).
- Efficient sampling: deterministic ODE solvers (DDIM, DPM-Solver) with 5–20 steps vs. 1000 steps for DDPM.
- Exact inversion: run the ODE forward to encode $\mathbf{x}_0 \rightarrow \mathbf{x}_T$, then backward to decode.

EXAMPLE — SDE vs. ODE Sampling for CIFAR-10 (Score-based Model)

Sampler	NFE	FID	Diversity
DDPM (ancestral, SDE)	1000	3.21	High
DDIM (ODE, $\eta = 0$)	50	4.04	Low
DPM-Solver-3 (ODE)	15	3.44	Medium
DPM-Solver++ (ODE)	10	3.12	Medium

NFE = number of function evaluations (score network queries). DPM-Solver++ achieves better FID than DDPM with $100\times$ fewer evaluations, by using a 3rd-order ODE solver that exploits the semi-linear structure of the probability flow ODE. The stochastic samplers (DDPM) produce more diverse samples; deterministic ODE samplers are more consistent (same noise $\rightarrow$ same output) but less diverse.

Pontryagin Maximum Principle and HJB Equation

Optimal control theory underpins many modern training algorithms. We derive the Pontryagin Maximum Principle and the Hamilton-Jacobi-Bellman equation, connecting neural ODE training to classical control.

48.1 Optimal Control Formulation of Deep Learning

Deep learning can be cast as a continuous-time optimal control problem: find control inputs (weight updates) to drive a dynamical system (neural network) from an initial state (random initialisation) to a target state (low-loss solution) while minimising a running cost.

Layer-as-time correspondence. An L -layer network with residual connections:

$$\mathbf{h}^{(l+1)} = \mathbf{h}^{(l)} + \frac{1}{L} f_{\theta^{(l)}}(\mathbf{h}^{(l)})$$

Under the substitution $t = l/L \in [0, 1]$, this is an Euler discretisation of:

$$\dot{\mathbf{h}}(t) = f_{\theta(t)}(\mathbf{h}(t)), \quad \mathbf{h}(0) = \mathbf{x}$$

The parameters $\theta(t)$ are the *control inputs* at time t (depth), and $\mathbf{h}(t)$ is the *state* (representation) at depth t .

Optimal control problem. Minimise the terminal cost (loss):

$$\min_{\theta(\cdot)} \phi(\mathbf{h}(1)) + \int_0^1 r(\theta(t), t) dt$$

subject to the state equation above, where $r(\theta, t)$ is a regularisation running cost (e.g., $\lambda \|\theta(t)\|^2$).

Why this formulation is useful.

- It unifies supervised learning (terminal cost) and regularisation (running cost) in a single framework.
- The PMP (§48.2) provides necessary conditions for optimality, showing that backpropagation computes the costate (adjoint) of this system.
- Continuous analysis enables stability results: a network is “well-posed” if the ODE has unique solutions, which is guaranteed when f_{θ} is Lipschitz in $\mathbf{h}$.
- Memory Networks and Neural ODEs are special cases: fixed θ (time-invariant control) corresponds to weight-tied networks; learned $\theta(t)$ corresponds to standard residual networks in the continuous limit.

EXAMPLE — ResNet-110 as Optimal Control

ResNet-110 on CIFAR-10: 110 layers, $\sim 1.7\text{M}$ parameters.

ODE interpretation: $t \in [0, 1]$ with $L = 55$ time steps of Euler integration (each residual block = 2 steps).

State dimension: $d_h = 64$ (feature maps after first convolution).

Control: $\theta(t)$ = two convolutional layers at each step, $\approx 30,000$ parameters per step.

Total control parameters: $55 \times 30000 = 1.65\text{M}$ — consistent with the actual parameter count.

The optimal control view suggests: ResNets are learning to transport the input distribution $p(\mathbf{x})$ to a linearly separable distribution $p(\mathbf{h}(1))$ in 55 Euler steps.

The connection between deep learning and optimal control, developed by E et al. (2017) and Li & Hao (2018), provides a continuous-time perspective on deep networks and reveals deep connections to dynamic programming, the adjoint method, and reinforcement learning.

Deep networks as dynamical systems. A deep residual network with L layers computes:

$$\mathbf{h}_{l+1} = \mathbf{h}_l + \frac{1}{L} f(\mathbf{h}_l, \square_l), \quad l = 0, 1, \dots, L-1$$

As $L \rightarrow \infty$, this converges to a controlled ODE:

$$\dot{\mathbf{h}}(t) = f(\mathbf{h}(t), \square(t)), \quad \mathbf{h}(0) = \mathbf{x}$$

The parameters $\square(t)$ are now a *control* — a time-varying policy that shapes the evolution of the hidden state.

Optimal Control Problem for Deep Learning

$$\min_{\square(\cdot)} \Phi(\mathbf{h}(T)) + \int_0^T L(\mathbf{h}(t), \square(t)) dt$$

subject to $\dot{\mathbf{h}} = f(\mathbf{h}, \square)$, $\mathbf{h}(0) = \mathbf{x}$.

- $\Phi(\mathbf{h}(T))$: terminal cost (classification loss at final layer).
- $L(\mathbf{h}, \square)$: running cost (regularisation on weights, e.g., $\|\square\|^2$).
- $f(\mathbf{h}, \square)$: dynamics (residual block function).

The Pontryagin Maximum Principle. The necessary conditions for optimality (see §48.2) give the backpropagation algorithm as a special case of the adjoint state method. This connection explains why backpropagation works: it computes the gradient of the loss with respect to all parameters simultaneously in a single backward pass, mirroring the co-state equations of optimal control.

Memory-efficient training via the adjoint method. The standard backpropagation algorithm requires storing all intermediate activations $\{\mathbf{h}(t)\}_{t \in [0, T]}$ during the forward pass, consuming $O(L)$ memory. The adjoint method (Chen et al. 2018, NeuralODE) solves the adjoint ODE backward in time:

$$\frac{d\mathbf{p}}{dt} = -\mathbf{p}^\top \frac{\partial f}{\partial \mathbf{h}}, \quad \mathbf{p}(T) = \frac{\partial \Phi}{\partial \mathbf{h}(T)}$$

recomputing $\mathbf{h}(t)$ as needed during the backward pass. This reduces memory to $O(1)$ at the cost of additional forward passes.

Connections to reinforcement learning. The optimal control formulation maps directly to RL:

- State: $\mathbf{h}(t)$ (hidden representation).
- Action: $\square(t)$ (parameters of the block).
- Reward: $-\Phi(\mathbf{h}(T))$ (negative classification loss).
- Policy: a meta-network that generates $\square(t)$ from $\mathbf{h}(t)$ (hypernetwork).

EXAMPLE — Memory Comparison: Backprop vs. Adjoint Method

For a Neural ODE with $d = 1024$, $T = 10$ (equivalent depth), batch size 32:

- Standard backprop (checkpointing every $k = 5$ steps): $O(d \times T/k) = O(2048)$ memory.
- Adjoint method: $O(d) = O(1024)$ memory (only current state + co-state).
- Adjoint overhead: $\approx 2\times$ compute (extra forward pass during backward).

The adjoint method enables training of very deep (large T) continuous networks at the cost of extra compute.

Training a neural ODE is a continuous-time optimal control problem:

$$\min_{\mathbf{u}(t)} \int_0^T r(\mathbf{h}(t), \mathbf{u}(t)) dt + \phi(\mathbf{h}(T)) \quad \text{s.t.} \quad \dot{\mathbf{h}} = f(\mathbf{h}, \mathbf{u}), \quad \mathbf{h}(0) = \mathbf{x}$$

where $\mathbf{u}(t)$ are the parameters (or learning rates), r is a running cost, and ϕ a terminal cost (the loss).

48.2 Pontryagin Maximum Principle

Define the *Hamiltonian*:

$$\mathcal{H}(\mathbf{h}, \mathbf{p}, \mathbf{u}) = \mathbf{p}^\top f(\mathbf{h}, \mathbf{u}) + r(\mathbf{h}, \mathbf{u})$$

where $\mathbf{p}(t)$ is the costate (adjoint) variable.

PMP Necessary Conditions

At the optimal control $\mathbf{u}^*(t)$, there exists a costate $\mathbf{p}^*(t)$ such that:

$$\dot{\mathbf{h}}^* = \frac{\partial \mathcal{H}}{\partial \mathbf{p}} = f(\mathbf{h}^*, \mathbf{u}^*) \quad (\text{state equation})$$

$$\dot{\mathbf{p}}^* = -\frac{\partial \mathcal{H}}{\partial \mathbf{h}} \quad (\text{costate/adjoint equation})$$

$$\mathbf{u}^*(t) = \arg \min_{\mathbf{u}} \mathcal{H}(\mathbf{h}^*, \mathbf{p}^*, \mathbf{u}) \quad (\text{optimality condition})$$

with boundary condition $\mathbf{p}^*(T) = \nabla_{\mathbf{h}} \phi(\mathbf{h}^*(T))$.

Connection to adjoint method. The costate ODE $\dot{\mathbf{p}}^* = -\frac{\partial \mathcal{H}}{\partial \mathbf{h}}$ is exactly the adjoint ODE from Part 47. The adjoint method is the PMP applied to neural ODE training.

48.2b Pontryagin Maximum Principle Applied to Deep Learning

The Pontryagin Maximum Principle (PMP) provides a necessary condition for optimality that is the continuous-time analogue of backpropagation. Understanding the PMP as applied to deep networks reveals why gradient descent works and what happens in the optimal limit.

Deep network as a dynamical system. An L -layer residual network:

$$\mathbf{h}^{(l+1)} = \mathbf{h}^{(l)} + \frac{1}{L} f_{\theta^{(l)}}(\mathbf{h}^{(l)})$$

As $L \rightarrow \infty$, this converges to the ODE $\dot{\mathbf{h}}(t) = f_\theta(\mathbf{h}(t), t)$ (time indexed by layer depth $t \in [0, 1]$). Minimising $\mathcal{L} = \phi(\mathbf{h}(1))$ subject to the ODE is an optimal control problem with:

- **State:** $\mathbf{h}(t)$ (hidden representation at depth t)
- **Control:** $\theta(t)$ (layer parameters at depth t)
- **Terminal cost:** $\phi(\mathbf{h}(1))$ (loss at output layer)

PMP necessary conditions. At an optimal control $\theta^*(t)$, there exists a costate $\mathbf{p}(t)$ satisfying:

$$\dot{\mathbf{h}}(t) = \frac{\partial H}{\partial \mathbf{p}} = f_{\theta^*(t)}(\mathbf{h}(t), t) \quad (\text{forward ODE})$$

$$\dot{\mathbf{p}}(t) = -\frac{\partial H}{\partial \mathbf{h}} = -\mathbf{p}(t)^\top \frac{\partial f_{\theta^*}}{\partial \mathbf{h}} \quad (\text{backward adjoint ODE})$$

$$\theta^*(t) = \arg \max_{\theta} H(\mathbf{h}(t), \mathbf{p}(t), \theta, t) \quad (\text{maximality condition})$$

where the Hamiltonian is $H = \mathbf{p}(t)^\top f_\theta(\mathbf{h}(t), t)$ and boundary condition $\mathbf{p}(1) = -\nabla_{\mathbf{h}} \phi(\mathbf{h}(1))$.

Connection to backpropagation. The costate equation $\dot{\mathbf{p}} = -\mathbf{p}^\top \partial f / \partial \mathbf{h}$ is precisely the adjoint ODE from §47.2. The adjoint $\mathbf{a}(t) = -\mathbf{p}(t)$ is the gradient $\partial \mathcal{L} / \partial \mathbf{h}(t)$. Backpropagation computes the adjoint/costate exactly.

The PMP maximality condition requires $\theta^*(t) = \arg \max_{\theta} H$, which is a *global* maximisation over the control θ at each time t . Gradient descent on θ finds only a local maximum of H . The gap between gradient-based training and PMP optimality represents the suboptimality introduced by non-convexity: deep networks trained by SGD satisfy the first-order necessary condition $\nabla_{\theta} H = 0$ (stationarity) but not necessarily the global maximality condition.

48.3 Hamilton-Jacobi-Bellman Equation

The Hamilton-Jacobi-Bellman (HJB) equation is the necessary and sufficient condition for optimality in continuous-time optimal control. It generalises the PMP and provides the theoretical foundation for reinforcement learning in continuous action spaces.

Value function. For a system $\dot{\mathbf{x}} = f(\mathbf{x}, \mathbf{u}, t)$ with running cost $r(\mathbf{x}, \mathbf{u})$ and terminal cost $\phi(\mathbf{x}(T))$, the value function is:

$$V(\mathbf{x}, t) = \min_{\mathbf{u}(\cdot)} \left[\int_t^T r(\mathbf{x}(s), \mathbf{u}(s)) ds + \phi(\mathbf{x}(T)) \right]$$

the minimum cost-to-go from state $\mathbf{x}$ at time t .

Hamilton-Jacobi-Bellman Equation

The value function satisfies the PDE:

$$-\frac{\partial V}{\partial t}(\mathbf{x}, t) = \min_{\mathbf{u}} [r(\mathbf{x}, \mathbf{u}) + \nabla_{\mathbf{x}} V(\mathbf{x}, t)^\top f(\mathbf{x}, \mathbf{u}, t)]$$

with terminal condition $V(\mathbf{x}, T) = \phi(\mathbf{x})$.

Derivation via dynamic programming. From the Bellman principle of optimality: $V(\mathbf{x}, t) = \min_{\mathbf{u}} [r(\mathbf{x}, \mathbf{u}) dt + V(\mathbf{x} + f dt, t + dt)]$.

Taylor-expanding $V(\mathbf{x} + f dt, t + dt) \approx V(\mathbf{x}, t) + \nabla_{\mathbf{x}} V^\top f dt + V_t dt$ and dividing by dt :

$$0 = \min_{\mathbf{u}} [r(\mathbf{x}, \mathbf{u}) + \nabla_{\mathbf{x}} V^\top f] + \frac{\partial V}{\partial t}$$

Rearranging gives the HJB.

Connection to Bellman equation in RL. The discrete-time Q-function $Q^\pi(s, a)$ satisfies the Bellman equation:

$$Q^\pi(s, a) = r(s, a) + \gamma \mathbb{E}_{s' \sim P(\cdot | s, a)} [V^\pi(s')]$$

As the time step $\Delta t \rightarrow 0$, $\gamma = e^{-\rho \Delta t}$, and the Bellman equation converges to the HJB. Continuous-time RL (e.g., soft actor-critic with continuous dynamics) directly optimises the HJB value function.

LQR: the linear-quadratic regulator. For linear dynamics $\dot{\mathbf{x}} = \mathbf{A}\mathbf{x} + \mathbf{B}\mathbf{u}$ and quadratic cost $r = \mathbf{x}^\top \mathbf{Q}\mathbf{x} + \mathbf{u}^\top \mathbf{R}\mathbf{u}$, the HJB has a closed-form solution $V(\mathbf{x}, t) = \mathbf{x}^\top \mathbf{P}(t)\mathbf{x}$ where $\mathbf{P}(t)$ satisfies the Riccati ODE:

$$-\dot{\mathbf{P}} = \mathbf{A}^\top \mathbf{P} + \mathbf{P}\mathbf{A} - \mathbf{P}\mathbf{B}\mathbf{R}^{-1}\mathbf{B}^\top \mathbf{P} + \mathbf{Q}$$

The optimal control is $\mathbf{u}^* = -\mathbf{R}^{-1}\mathbf{B}^\top \mathbf{P}\mathbf{x}$ (linear feedback). The Riccati equation is the continuous-time analogue of the backward Bellman recursion in discrete RL.

EXAMPLE — HJB Applied to Neural ODE Training

Neural ODE training = optimal control of $\dot{\mathbf{h}} = f_\theta(\mathbf{h}, t)$.

Value function: $V(\mathbf{h}, t) = \min_{\theta(\cdot)} \mathcal{L}(\mathbf{h}(T); \theta(\cdot)) =$ minimum loss achievable from state $\mathbf{h}$ at depth t .

HJB: $\partial V / \partial t = -\min_{\theta} [\nabla_{\mathbf{h}} V^\top f_\theta(\mathbf{h}, t)]$.

The gradient $\nabla_{\mathbf{h}} V = \mathbf{a}(t)$ is the adjoint (§47.2). The HJB says: at each depth t , choose $\theta(t)$ to minimally decrease the value — i.e., choose the control that best reduces $\nabla_{\mathbf{h}} V^\top f_\theta$.

This is precisely what gradient descent on θ accomplishes: $\theta(t) \leftarrow \theta(t) - \eta \nabla_{\theta} (\nabla_{\mathbf{h}} V^\top f_\theta)$. Training a Neural ODE is solving the HJB approximately via gradient descent.

Define the value function $V(\mathbf{h}, t) = \min_{\mathbf{u}} \int_t^T r ds + \phi(\mathbf{h}(T))$. It satisfies the HJB PDE:

$$-\frac{\partial V}{\partial t} = \min_{\mathbf{u}} [r(\mathbf{h}, \mathbf{u}) + (\nabla_{\mathbf{h}} V)^\top f(\mathbf{h}, \mathbf{u})]$$

with terminal condition $V(\mathbf{h}, T) = \phi(\mathbf{h})$.

The HJB equation is a nonlinear PDE. For linear-quadratic systems it reduces to the Riccati equation; for nonlinear systems, approximate dynamic programming and deep RL methods are used.

48.4 Stochastic Optimal Control and Policy Gradient Methods

The connection between optimal control and reinforcement learning becomes explicit in the stochastic setting, where the state dynamics involve noise and the control objective is to minimise expected cost.

Stochastic optimal control. Consider the controlled stochastic differential equation:

$$d\mathbf{x}(t) = f(\mathbf{x}(t), \mathbf{u}(t)) dt + \sigma(\mathbf{x}(t)) d\mathbf{W}(t)$$

where $\mathbf{W}(t)$ is a standard Wiener process (Brownian motion) and σ is the diffusion coefficient. The stochastic value function:

$$V(\mathbf{x}, t) = \min_{\mathbf{u}(\cdot)} \mathbb{E} \left[\int_t^T r(\mathbf{x}(s), \mathbf{u}(s)) ds + \phi(\mathbf{x}(T)) \mid \mathbf{x}(t) = \mathbf{x} \right]$$

satisfies the stochastic HJB equation (Bellman's principle for diffusions):

$$-\frac{\partial V}{\partial t} = \min_{\mathbf{u}} \left[r(\mathbf{x}, \mathbf{u}) + \nabla_{\mathbf{x}} V^\top f(\mathbf{x}, \mathbf{u}) + \frac{1}{2} \text{tr}(\sigma \sigma^\top \nabla_{\mathbf{x}}^2 V) \right]$$

The additional term $\frac{1}{2} \text{tr}(\sigma \sigma^\top \nabla_{\mathbf{x}}^2 V)$ is the Itô correction, arising from the quadratic variation of the Brownian motion. This term acts as a “diffusion regulariser”: the optimal policy accounts for the noise by smoothing the value function.

Policy gradient (REINFORCE). In model-free RL, we parameterise a stochastic policy $\pi_\theta(\mathbf{u}|\mathbf{x})$ and optimise the expected return:

$$J(\theta) = \mathbb{E}_{\tau \sim \pi_\theta} \left[\sum_t r(\mathbf{x}_t, \mathbf{u}_t) \right]$$

The policy gradient theorem (Williams 1992) gives:

$$\nabla_\theta J(\theta) = \mathbb{E}_{\tau \sim \pi_\theta} \left[\sum_t \nabla_\theta \log \pi_\theta(\mathbf{u}_t | \mathbf{x}_t) \cdot G_t \right]$$

where $G_t = \sum_{k \geq t} r_k$ is the return from step t . This is the *REINFORCE* gradient estimator: the score function estimator applied to the policy.

Connection to the adjoint method. The policy gradient is the discrete-time analogue of the adjoint method. Define the costate $\mathbf{p}_t = \partial J / \partial \mathbf{x}_t$ (value function gradient). The backward recursion:

$$\mathbf{p}_t = r_t + \gamma \mathbb{E}_{\mathbf{x}_{t+1} | \mathbf{x}_t, \mathbf{u}_t} [\mathbf{J}_f^\top \mathbf{p}_{t+1}]$$

mirrors the adjoint ODE from §47.2: $\dot{\mathbf{a}} = -\mathbf{a}^\top \partial f / \partial \mathbf{h}$. The Bellman recursion is the backward-in-time costate equation, evaluated at discrete time steps.

Variance reduction for REINFORCE. The REINFORCE gradient has high variance: the return G_t fluctuates greatly across trajectories. The baseline trick subtracts a *baseline* $b(\mathbf{x}_t)$ (independent of the action) without introducing bias:

$$\nabla_\theta J(\theta) = \mathbb{E} \left[\sum_t \nabla_\theta \log \pi_\theta(\mathbf{u}_t | \mathbf{x}_t) \cdot (G_t - b(\mathbf{x}_t)) \right]$$

The optimal baseline is $b^*(\mathbf{x}) = \mathbb{E}[G_t | \mathbf{x}_t = \mathbf{x}] = V^\pi(\mathbf{x})$ (the value function). Learning $b \approx V^\pi$ alongside the policy is the actor-critic architecture:

- *Actor*: $\pi_\theta(\mathbf{u}|\mathbf{x})$ — the policy (control).
- *Critic*: $V_\psi(\mathbf{x})$ — the value function (costate approximation).
- *Advantage*: $A_t = G_t - V_\psi(\mathbf{x}_t)$ — the baseline-corrected return.

EXAMPLE — Advantage Estimation: GAE

Generalised Advantage Estimation (GAE, Schulman et al. 2016) balances bias and variance in the advantage estimate via a weighting parameter $\lambda \in [0, 1]$:

$$\hat{A}_t^{\text{GAE}} = \sum_{k=0}^{\infty} (\gamma\lambda)^k \delta_{t+k}$$

where $\delta_t = r_t + \gamma V(\mathbf{x}_{t+1}) - V(\mathbf{x}_t)$ is the temporal difference (TD) error.

- $\lambda = 0$: $\hat{A}_t = \delta_t$ (TD(0), low variance, high bias from V errors).
- $\lambda = 1$: $\hat{A}_t = G_t - V(\mathbf{x}_t)$ (Monte Carlo, zero bias, high variance).
- $\lambda = 0.95$ (PPO default): exponentially-weighted sum, trades off bias and variance.

GAE is the standard advantage estimator in modern policy gradient methods (PPO, A3C, RLHF).

Proximal Policy Optimisation (PPO). The policy gradient suffers from large step sizes that destabilise training. PPO clips the policy update to remain within a trust region:

$$\mathcal{L}_{\text{PPO}}(\theta) = \mathbb{E}_t \left[\min \left(r_t(\theta) \hat{A}_t, \text{clip}(r_t(\theta), 1 - \varepsilon, 1 + \varepsilon) \hat{A}_t \right) \right]$$

where $r_t(\theta) = \pi_\theta(\mathbf{u}_t | \mathbf{x}_t) / \pi_{\theta_{\text{old}}}(\mathbf{u}_t | \mathbf{x}_t)$ is the probability ratio. The clip operation prevents large policy updates while the min ensures the estimate is always pessimistic (prevents overestimation of improvement). PPO is the dominant RL algorithm for RLHF (Reinforcement Learning from Human Feedback) in large language models.

48.5 Hamiltonian Monte Carlo and Variational Inference

Hamiltonian Monte Carlo (HMC) uses the Hamiltonian mechanics framework of §48.2 for Bayesian neural network inference, bridging gradient-based optimisation and posterior sampling.

Bayesian neural networks via HMC. Rather than finding the MAP estimate $\theta^* = \arg \max_\theta \log p(\theta | \mathcal{D})$, Bayesian inference computes the full posterior $p(\theta | \mathcal{D}) \propto p(\mathcal{D} | \theta) p(\theta)$. HMC samples from this posterior by treating $-\log p(\theta | \mathcal{D})$ as the potential energy and augmenting with momentum $\mathbf{m}$:

$$H(\theta, \mathbf{m}) = -\log p(\theta | \mathcal{D}) + \frac{1}{2} \mathbf{m}^\top \mathbf{M}^{-1} \mathbf{m}$$

Leapfrog integration follows the Hamiltonian dynamics:

$$\mathbf{m}_{t+\epsilon/2} = \mathbf{m}_t - \frac{\epsilon}{2} \nabla_\theta U(\theta_t)$$

$$\theta_{t+\epsilon} = \theta_t + \epsilon \mathbf{M}^{-1} \mathbf{m}_{t+\epsilon/2}$$

$$\mathbf{m}_{t+\epsilon} = \mathbf{m}_{t+\epsilon/2} - \frac{\epsilon}{2} \nabla_\theta U(\theta_{t+\epsilon})$$

After L leapfrog steps, the proposal $(\theta', \mathbf{m}')$ is accepted with Metropolis probability $\min(1, e^{-H(\theta', \mathbf{m}') + H(\theta, \mathbf{m})})$.

Mass matrix preconditioning. The mass matrix $\mathbf{M}$ plays the same role as the preconditioner in natural gradient descent: it sets the scale of each parameter dimension. Optimal choice: $\mathbf{M} = \mathbf{F}(\theta)^{-1}$ (inverse Fisher). This makes the leapfrog dynamics isotropic (equal step sizes in all natural directions) and dramatically improves mixing. In practice, a diagonal approximation $\mathbf{M} = \text{diag}(\mathbf{F})^{-1}$ is used.

Stochastic gradient HMC (SGHMC). For large datasets, computing $\nabla_{\theta} U(\theta) = -\nabla_{\theta} \log p(\theta|\mathcal{D})$ requires the full dataset. SGHMC (Chen et al., 2014) uses mini-batch gradient estimates with a friction term:

$$d\mathbf{m} = -\nabla_{\theta} \tilde{U}(\theta) dt - C\mathbf{M}^{-1}\mathbf{m} dt + \sqrt{2C} d\mathbf{W}$$

The friction term $C\mathbf{M}^{-1}\mathbf{m}$ damps the momentum (preventing runaway from gradient noise) and the diffusion term $\sqrt{2C} d\mathbf{W}$ injects the correct amount of noise to maintain the target posterior. This is the Langevin dynamics from §40.2 with momentum.

Connection to SGD. Standard SGD with learning rate η is equivalent to SGHMC with:

- Friction $C = 1$ (overdamped, momentum decays in one step).
- Temperature $T = \sigma_{\text{noise}}^2/2$ (gradient noise determines temperature).
- Mass matrix $\mathbf{M} = \mathbf{I}/\eta$ (step size sets mass).

SGD implicitly samples from a distribution around the MAP estimate, not the MAP estimate itself. For large η , the implicit temperature is high (broad distribution, poor fit); for small η , the temperature is low (distribution concentrates at MAP). Adam’s adaptive step sizes set different temperatures per dimension, corresponding to a diagonal non-isotropic distribution.

EXAMPLE — HMC vs. Variational Inference for BNN on MNIST

2-layer BNN, 100 hidden units, MNIST, prior $\mathcal{N}(0, \mathbf{I})$:

Method	Test Acc	ECE (calibration)	Cost
MAP (standard training)	98.4%	0.028	1×
MC Dropout	98.4%	0.021	2×
Mean-field VI (ELBO)	98.3%	0.015	3×
SGHMC (100 samples)	98.6%	0.009	50×

ECE = Expected Calibration Error (lower is better). SGHMC achieves the best calibration at high computational cost. Mean-field VI offers a good accuracy-calibration-cost tradeoff. MC Dropout is the cheapest approximate Bayesian method.

Signal Processing Foundations: Nyquist-Shannon and Aliasing

Convolutional neural networks process discrete signals. We derive the sampling theorem, aliasing, and anti-aliasing, and show how equivariance arguments motivate blur-pooling in modern CNNs.

49.1 Continuous-Time Fourier Transform

The Fourier transform decomposes a signal into sinusoidal components of different frequencies. It is the foundation of signal processing and connects spatial operations (convolution) to spectral operations (multiplication).

Continuous Fourier Transform

For $f \in L^1(\mathbb{R})$ (or $L^2(\mathbb{R})$):

$$\hat{f}(\xi) = \int_{-\infty}^{\infty} f(t) e^{-2\pi i \xi t} dt$$

The inverse Fourier transform:

$$f(t) = \int_{-\infty}^{\infty} \hat{f}(\xi) e^{2\pi i \xi t} d\xi$$

Key properties.

- *Linearity*: $\widehat{af + bg} = a\hat{f} + b\hat{g}$.
- *Convolution theorem*: $\widehat{f * g} = \hat{f} \cdot \hat{g}$.
- *Shift*: $\widehat{f(t - t_0)}(\xi) = e^{-2\pi i \xi t_0} \hat{f}(\xi)$ (shift in time = phase shift in frequency).
- *Derivative*: $\widehat{f'}(\xi) = 2\pi i \xi \hat{f}(\xi)$ (differentiation = multiplication by $2\pi i \xi$).
- *Parseval*: $\int |f(t)|^2 dt = \int |\hat{f}(\xi)|^2 d\xi$ (energy is conserved).
- *Uncertainty principle*: $\sigma_t \cdot \sigma_\xi \geq 1/(4\pi)$ where σ_t, σ_ξ are the widths of f and $\hat{f}$ respectively. A narrow pulse in time has broad frequency content (and vice versa).

The Gaussian and its Fourier transform. A Gaussian $f(t) = e^{-\pi t^2}$ satisfies $\hat{f}(\xi) = e^{-\pi \xi^2}$: the Fourier transform of a Gaussian is a Gaussian. This is the fixed point of the Fourier transform and the reason Gaussian blur (§49.3) is ideal for low-pass filtering: the Gaussian is its own spectral representation.

EXAMPLE — Fourier Transform of a Rectangular Pulse

$f(t) = \mathbf{1}[|t| \leq W/2]$ (pulse of width W).

$$\hat{f}(\xi) = \int_{-W/2}^{W/2} e^{-2\pi i \xi t} dt = W \cdot \text{sinc}(W\xi), \quad \text{sinc}(x) = \frac{\sin(\pi x)}{\pi x}$$

A pulse of width W in time has frequency content spread over $O(1/W)$ bandwidth. For a 3×3 convolution kernel (spatial extent $W = 3$ pixels): frequency content up to $\xi = 1/3$ cycles/pixel. The Nyquist frequency for stride-2 sampling is $1/4$ cycles/pixel.

Since $1/3 > 1/4$, the 3×3 kernel contains frequencies above Nyquist and will alias when stride-2 subsampled without blurring.

The Fourier transform decomposes a signal into its frequency components, providing a dual representation that makes many operations (convolution, differentiation, filtering) algebraically transparent. Its discretisation underlies the computational engines of CNNs, audio processing, and spectral graph methods.

Definition. For $f \in L^1(\mathbb{R})$, the Fourier transform and its inverse are:

$$\hat{f}(\xi) = \int_{-\infty}^{\infty} f(t) e^{-2\pi i \xi t} dt, \quad f(t) = \int_{-\infty}^{\infty} \hat{f}(\xi) e^{2\pi i \xi t} d\xi$$

For $f \in L^2(\mathbb{R})$, the transform extends by continuity and preserves the L^2 norm: $\|f\|_2 = \|\hat{f}\|_2$ (Parseval's theorem).

Key Fourier Properties

- **Convolution theorem:** $\widehat{f * g}(\xi) = \hat{f}(\xi) \hat{g}(\xi)$. Convolution in time = pointwise multiplication in frequency.
- **Differentiation:** $\widehat{f'}(\xi) = 2\pi i \xi \hat{f}(\xi)$. Derivatives become multiplications.
- **Shift:** $\widehat{f(\cdot - t_0)}(\xi) = e^{-2\pi i \xi t_0} \hat{f}(\xi)$. Time shift = phase shift.
- **Scaling:** $\widehat{f(at)}(\xi) = \frac{1}{|a|} \hat{f}(\xi/a)$. Compression in time = dilation in frequency.

Convolution theorem and CNNs. The convolution theorem explains why CNNs can be implemented efficiently in the frequency domain. A 2D convolution $(\mathbf{f} * \mathbf{k})[\mathbf{n}]$ with kernel $\mathbf{k}$ can be computed as:

$$\text{conv}(\mathbf{f}, \mathbf{k}) = \mathcal{F}^{-1}[\mathcal{F}[\mathbf{f}] \odot \mathcal{F}[\mathbf{k}]]$$

using the 2D DFT/IDFT. For an image of size $N \times N$ and kernel size $K \times K$:

- Direct convolution: $O(N^2 K^2)$.
- FFT-based convolution: $O(N^2 \log N)$ for large kernels ($K \gg \log N$).

In practice, FFT-based convolution is preferred for large kernels (e.g., non-local attention) but direct computation dominates for the small kernels (3×3 , 5×5) used in standard CNNs.

Spectral bias of neural networks. Neural networks trained with gradient descent exhibit a *spectral bias* (also called *frequency principle*): they learn low-frequency components of the target function before high-frequency ones. This is captured by the neural tangent kernel: the NTK eigenvalues decay with frequency, causing low-frequency modes to converge faster. Concretely, for a two-layer network fitting a target $f(x) = \sum_k c_k e^{2\pi i k x}$, the k -th Fourier coefficient is learned at rate $\sim |k|^{-2}$, explaining why neural networks are biased toward smooth functions.

EXAMPLE — Fourier Features for Positional Encoding

Transformer positional encoding uses Fourier features:

$$\text{PE}(t, 2j) = \sin\left(\frac{t}{10000^{2j/d}}\right), \quad \text{PE}(t, 2j+1) = \cos\left(\frac{t}{10000^{2j/d}}\right)$$

The base $10000^{2j/d}$ sets the frequency for dimension j , ranging from $1/(10000)$ for

$j = d/2 - 1$ (slow oscillation) to 1 for $j = 0$ (fast oscillation). This multi-frequency encoding allows the model to represent positions at all scales simultaneously, supporting both local (high-frequency) and global (low-frequency) positional relationships.

Short-time Fourier transform and wavelets. For non-stationary signals, the global Fourier transform loses temporal localisation. The short-time Fourier transform (STFT) windowed at time t is $\mathcal{S}f(t, \xi) = \int f(\tau)g(\tau - t)e^{-2\pi i \xi \tau} d\tau$. Wavelets replace the fixed window with a scaled mother wavelet ψ : $\mathcal{W}f(a, b) = \frac{1}{\sqrt{a}} \int f(t)\psi^*((t - b)/a)dt$, providing adaptive time-frequency resolution.

For $f \in L^1(\mathbb{R})$:

$$\hat{f}(\xi) = \int_{-\infty}^{\infty} f(t)e^{-2\pi i \xi t} dt, \quad f(t) = \int_{-\infty}^{\infty} \hat{f}(\xi)e^{2\pi i \xi t} d\xi$$

The Fourier transform converts convolution to pointwise multiplication: $\widehat{f * g}(\xi) = \hat{f}(\xi)\hat{g}(\xi)$.

49.2 Nyquist-Shannon Sampling Theorem

Nyquist-Shannon Theorem

A signal $f(t)$ bandlimited to $[-B, B]$ Hz (i.e. $\hat{f}(\xi) = 0$ for $|\xi| > B$) can be perfectly reconstructed from its samples $\{f(n/f_s)\}_{n \in \mathbb{Z}}$ if and only if the sampling rate $f_s > 2B$ (the Nyquist rate).

Reconstruction formula:

$$f(t) = \sum_{n=-\infty}^{\infty} f\left(\frac{n}{f_s}\right) \cdot \text{sinc}(f_s t - n), \quad \text{sinc}(x) = \frac{\sin(\pi x)}{\pi x}$$

Proof via Poisson summation. Sampling at rate f_s is equivalent to multiplying by a Dirac comb $\sum_n \delta(t - n/f_s)$. In the frequency domain, this convolves $\hat{f}$ with a comb of spacing f_s , creating periodic copies of $\hat{f}$. If $f_s > 2B$, the copies do not overlap (no aliasing), and a brick-wall lowpass filter of bandwidth $f_s/2$ recovers $\hat{f}$ exactly. If $f_s \leq 2B$, the copies overlap and aliasing occurs: irreversible spectral folding.

49.3 Aliasing in CNNs and Blur-Pooling

CNNs with strided convolutions or max-pooling violate the Nyquist theorem: they subsample the feature map without first low-pass filtering, creating aliasing artifacts.

The aliasing problem in max-pooling. Max-pooling with stride s computes:

$$y[n] = \max_{k \in \{0, \dots, K-1\}} x[sn + k]$$

The max operation is not band-limited: it can introduce high-frequency content. Subsampling by s then aliases these high frequencies back into the passband.

Practical consequence. Zhang (2019) demonstrated that standard CNNs are not shift-invariant: shifting an input by 1 pixel can change the output by $\sim 10\%$ of its magnitude, and can even flip the predicted class. This is because the aliased high-frequency content encodes absolute spatial position rather than spatial structure.

Blur-pooling. The fix is simple: apply a low-pass filter (blur) before each stride operation. For stride-2 pooling, use a Gaussian-like kernel before subsampling:

$$y[n] = \sum_k h[k] \cdot x'[2n - k], \quad x' = \text{MaxPool}_{1 \times 1}(x)$$

Standard blur kernels include $[1, 2, 1]/4$ (binomial-3) or $[1, 4, 6, 4, 1]/16$ (binomial-5). The blur removes frequencies above $f_s/4$ (the Nyquist frequency for stride-2), preventing aliasing.

EXAMPLE — Shift Invariance Before and After Blur-Pooling

ResNet-50 on ImageNet, measuring consistency: the fraction of test images where shifting the input by 1 pixel gives the same predicted class.

Architecture	Shift Consistency	Top-1 Accuracy
Standard ResNet-50	72.3%	76.1%
ResNet-50 + BlurPool	88.6%	76.4%

Blur-pooling increases shift consistency by 16% with a negligible accuracy change. The improvement comes without any additional parameters: blur kernels are fixed (not learned).

Standard max-pooling with stride 2 can alias: the max operation is not bandlimited, so downsampling by 2 aliases high-frequency content. Zhang (2019) proposed *blur-pooling*:

1. Apply a fixed low-pass filter (e.g. binomial filter $[1, 2, 1]/4$) to the feature map.
2. Then downsample by the stride.

This ensures shift-equivariance: shifting the input by one pixel shifts the output by one pixel (rather than causing arbitrary feature changes), which improves generalisation and robustness.

EXAMPLE — Binomial blur filter application

Feature map (1D): $\mathbf{h} = (2, 8, 3, 6, 1, 4)$.

Blur filter: $[1, 2, 1]/4$. After blurring:

$$h'_1 = (2 \cdot 1 + 8 \cdot 2 + 3 \cdot 1)/4 = 21/4 = 5.25$$

$$h'_2 = (8 \cdot 1 + 3 \cdot 2 + 6 \cdot 1)/4 = 20/4 = 5.0$$

$$h'_3 = (3 \cdot 1 + 6 \cdot 2 + 1 \cdot 1)/4 = 16/4 = 4.0$$

After stride-2 subsampling: $(5.25, 4.0)$ — smoother than direct stride-2 max of $(8, 6)$.

49.4 Spectral Analysis of Convolutional Architectures

The Fourier and Nyquist frameworks provide rigorous tools for analysing CNN design choices: pooling strategies, activation functions, and network depth.

Spectral view of max-pooling vs. average pooling. Average pooling with stride s is equivalent to:

1. Low-pass filter with impulse response $h[n] = \frac{1}{s} \mathbf{1}[0 \leq n < s]$.

2. Downsample by s .

The filter h has frequency response $\hat{h}(\xi) = \frac{1}{s} \sum_{k=0}^{s-1} e^{-2\pi i k \xi / s} = e^{-\pi i (s-1) \xi / s} \text{sinc}(s\xi)$. For stride $s = 2$: the filter passes frequencies below $1/(2s) = 1/4$ cycles/pixel and attenuates higher frequencies. This is anti-aliasing: average pooling is Nyquist-correct.

Max pooling selects the maximum over a window rather than the average. The max operator is not linear and not bandlimited: $\max(f, g) = \frac{f+g}{2} + \frac{|f-g|}{2}$, where $|f-g|$ introduces high-frequency content. Subsampling after max pooling therefore aliases.

Spectral norm of convolution as a Lipschitz bound. A convolutional layer $\mathbf{z} = \mathbf{W} * \mathbf{x}$ (circular convolution) has spectral norm:

$$\|\mathbf{W} * \cdot\|_2 = \max_{\xi} |\hat{W}(\xi)|$$

where $\hat{W}(\xi)$ is the DFT of the convolutional filter. This equals the maximum amplitude in the frequency domain. For a 3×3 filter, $\max_{\xi} |\hat{W}(\xi)|$ can be computed in $O(d^2)$ via FFT and used for spectral normalisation (§19.4 in Volume II) of convolutional layers.

Shift equivariance and spectral regularity. The shift equivariance of a CNN degrades when non-linearities (ReLU, max pooling) are inserted without anti-aliasing. In the frequency domain:

$$|\Phi(T_s f)(\xi) - T_s \Phi(f)(\xi)| \leq C_{\text{alias}} \cdot \epsilon_s$$

where ϵ_s is the aliasing error from subsampling and C_{alias} depends on the spectral content above the Nyquist frequency. Blur-pooling reduces ϵ_s to near-zero, restoring shift equivariance.

EXAMPLE — Frequency Analysis of ResNet Pooling Strategies

ResNet-50 input: 224×224 RGB image. Frequency range: $[0, 0.5]$ cycles/pixel (Nyquist at stride 1).

After stride-2 max-pool in the first layer (stride-2 7×7 conv + stride-2 max):

- Output: 56×56 feature map. Nyquist of output: 0.5 cyc/pixel relative to new resolution = 0.25 cyc/pixel of original.
- Aliased content: all original frequencies > 0.25 cyc/pixel are folded back.
- Max-pool introduces frequencies up to 0.5 cyc/pixel before subsampling — aliased by stride 2.

After stride-2 average-pool (blur-pool):

- The averaging filter $[1, 1]/2$ has frequency response $\cos(\pi\xi)$, attenuating frequencies near 0.5 cyc/pixel.
- Aliasing error $\approx \sum_{\xi > 0.25} |\hat{x}(\xi)|^2 \cdot |\hat{h}(0.5 - \xi)|^2 \approx 3\%$ of total power.

The difference: max-pool aliases $\sim 15\%$ of input power; average-pool aliases $\sim 3\%$. This $5\times$ reduction in aliasing directly translates to 16% improved shift consistency (Table in §49.3).

Depthwise separable convolutions and spectral efficiency. MobileNets (Howard et al., 2017) factorise a $K \times K \times C \times C'$ convolution into:

1. Depthwise: $K \times K \times 1$ (one filter per channel), cost $K^2 \cdot C \cdot H \cdot W$.
2. Pointwise: $1 \times 1 \times C \times C'$ (mix channels), cost $C \cdot C' \cdot H \cdot W$.

Total cost: $(K^2 + C') \cdot C \cdot H \cdot W$ vs. $K^2 \cdot C \cdot C' \cdot H \cdot W$ for standard convolution. Speedup factor: $\approx K^2 / (1 + K^2 / C')$. For $K = 3$, $C' = 256$: speedup $\approx 9\times$.

In the spectral domain: depthwise convolution applies independent linear filters per channel (no cross-channel mixing), while pointwise convolution applies cross-channel mixing without spatial filtering. The factorisation is spectrally lossless: the composed operator has the same frequency response as the original convolution.

Equivariance in Convolutions and the Sampling Theorem Revisited

We formalise translation equivariance of convolutions, derive the commutation relation, and explore how group-equivariant convolutions generalise the classical CNN to arbitrary symmetry groups.

50.1 Translation Equivariance of Continuous Convolution

Equivariance is the mathematical foundation of convolutional networks: the property that spatial transformations of the input produce corresponding transformations of the output.

Equivariance

A function $\Phi : \mathcal{X} \rightarrow \mathcal{Y}$ is *equivariant* with respect to a group action $\rho_g : \mathcal{X} \rightarrow \mathcal{X}$ (and $\rho'_g : \mathcal{Y} \rightarrow \mathcal{Y}$) if:

$$\Phi(\rho_g(x)) = \rho'_g(\Phi(x)) \quad \forall g \in G, x \in \mathcal{X}$$

In words: applying the group action before or after Φ gives the same result.

For translation ($g = \mathbf{t}$, $\rho_{\mathbf{t}}[f](\mathbf{x}) = f(\mathbf{x} - \mathbf{t})$), equivariance means:

$$\Phi(f(\cdot - \mathbf{t}))(\mathbf{x}) = \Phi(f)(\mathbf{x} - \mathbf{t})$$

Convolution is equivariant to translation. Let $(f * h)(\mathbf{x}) = \int f(\mathbf{y})h(\mathbf{x} - \mathbf{y})d\mathbf{y}$.

Proof. For a translated input $f_{\mathbf{t}}(\mathbf{x}) = f(\mathbf{x} - \mathbf{t})$:

$$\begin{aligned} (f_{\mathbf{t}} * h)(\mathbf{x}) &= \int f(\mathbf{y} - \mathbf{t})h(\mathbf{x} - \mathbf{y})d\mathbf{y} \\ &= \int f(\mathbf{u})h(\mathbf{x} - \mathbf{t} - \mathbf{u})d\mathbf{u} \quad (\mathbf{u} = \mathbf{y} - \mathbf{t}) \\ &= (f * h)(\mathbf{x} - \mathbf{t}) \end{aligned}$$

This is the translation-equivariance property: translating the input translates the output by the same amount. $\square$

Why equivariance matters for CNNs. In a CNN, the feature map $\Phi^{(l)}(\mathbf{x})$ at each layer is equivariant to translation. This means: if a cat is in the top-left of the image, the feature response “cat present” fires in the top-left of the feature map. If the cat is in the bottom-right, the same feature fires in the bottom-right. The convolution kernel does not need to “learn” the cat at every spatial position separately — it learns the cat once, and equivariance handles the rest.

Invariance via global pooling. A classifier needs invariance (not equivariance): the label “cat” should not depend on where the cat is. Global average pooling $p = \frac{1}{HW} \sum_{i,j} \Phi_{i,j}$ achieves this: since Φ is equivariant, global average pooling of the translated feature map equals global average pooling of the original:

$$\frac{1}{HW} \sum_{i,j} \Phi_{i,j}(f(\cdot - \mathbf{t})) = \frac{1}{HW} \sum_{i,j} \Phi_{i-t_1, j-t_2}(f) = \frac{1}{HW} \sum_{i,j} \Phi_{i,j}(f)$$

(assuming toroidal/periodic boundary; for finite images, with boundary effects).

Translation equivariance is exact only for continuous convolution and for discrete convolution *without* stride or pooling. Strided convolution and max-pooling break equivariance due to aliasing (§49.3). The standard solution: blur before stride (blur-pooling) and avoid max-pooling in favour of average pooling + stride.

Define the translation operator $[T_s f](t) = f(t - s)$. Convolution $[f * g](t) = \int f(\tau)g(t - \tau) d\tau$ is translation equivariant:

$$[(T_s f) * g](t) = \int f(\tau - s)g(t - \tau) d\tau = [f * g](t - s) = T_s[f * g](t)$$

So $T_s \circ (\cdot * g) = (\cdot * g) \circ T_s$: convolution commutes with translation.

50.2 Discrete Convolution and Circular Convolution

For discrete signals $\mathbf{f}, \mathbf{g} \in \mathbb{R}^N$, *circular convolution* is:

$$[\mathbf{f} \circledast \mathbf{g}]_n = \sum_{m=0}^{N-1} f_m g_{(n-m) \bmod N}$$

Equivalent via the DFT: $\mathcal{F}\{\mathbf{f} \circledast \mathbf{g}\} = \mathcal{F}\{\mathbf{f}\} \odot \mathcal{F}\{\mathbf{g}\}$ (pointwise).

Cyclic group equivariance. The cyclic group $\mathbb{Z}_N$ acts on $\mathbb{R}^N$ by cyclic shift. Circular convolution is $\mathbb{Z}_N$ -equivariant: shifting the input cyclically shifts the output cyclically.

50.2b The DFT Convolution Theorem and Efficient Computation

The Discrete Fourier Transform (DFT) converts circular convolution to pointwise multiplication, enabling $O(n \log n)$ convolution via the Fast Fourier Transform (FFT).

Circular convolution. For periodic signals x, h of length N :

$$(x \circledast h)[n] = \sum_{k=0}^{N-1} x[k] \cdot h[(n - k) \bmod N]$$

Convolution theorem. The DFT $\hat{x}[f] = \sum_{n=0}^{N-1} x[n]e^{-2\pi i f n/N}$ satisfies:

$$\widehat{x \circledast h}[f] = \hat{x}[f] \cdot \hat{h}[f]$$

Proof. $\sum_n (x \circledast h)[n]e^{-2\pi i f n/N} = \sum_n \sum_k x[k]h[n - k]e^{-2\pi i f n/N} = \hat{x}[f] \cdot \hat{h}[f]$ (after change of variable $m = n - k$, using $\sum_m h[m]e^{-2\pi i f m/N} = \hat{h}[f]$).

FFT-based convolution algorithm.

1. Zero-pad x and h to length $N \geq n_x + n_h - 1$ (to convert circular to linear convolution).
2. Compute $\hat{x} = \text{FFT}(x)$ and $\hat{h} = \text{FFT}(h)$ in $O(N \log N)$.
3. Pointwise multiply: $\hat{y}[f] = \hat{x}[f] \cdot \hat{h}[f]$.
4. Inverse FFT: $y = \text{IFFT}(\hat{y})$ in $O(N \log N)$.

Total cost: $O(N \log N)$ vs. $O(N^2)$ for direct convolution. For $N = 10^4$: 1.3×10^5 vs. 10^8 multiplications — a $770\times$ speedup.

Winograd convolution for CNNs. For small kernels ($3 \times 3, 5 \times 5$), Winograd (1980) provides minimal-multiplication algorithms. For 3×3 kernels on 2×2 output tiles ($F(2 \times 2, 3 \times 3)$): reduces multiplications from 36 to 16 by precomputing sums. PyTorch’s CUDNN backend automatically selects between FFT-based and Winograd algorithms based on kernel size and batch dimensions.

50.3 Steerable Filters and $SE(2)$ Equivariance

While CNNs are equivariant to translation, they are not equivariant to rotation. A cat oriented at 45° looks different to a standard CNN than a cat oriented at 0° . Steerable filters achieve equivariance to the full group $SE(2)$ = translations + rotations in 2D.

The $SE(2)$ group. Elements of $SE(2)$ are pairs $(r, \mathbf{t})$ with $r \in [0, 2\pi)$ (rotation angle) and $\mathbf{t} \in \mathbb{R}^2$ (translation). The group action on images:

$$(\rho_{r,\mathbf{t}}f)(\mathbf{x}) = f(\mathbf{R}_{-r}(\mathbf{x} - \mathbf{t})), \quad \mathbf{R}_{-r} = \begin{pmatrix} \cos r & \sin r \\ -\sin r & \cos r \end{pmatrix}$$

Steerable filter theory. A filter $h : \mathbb{R}^2 \rightarrow \mathbb{R}$ is *steerable* if its rotations can be expressed as linear combinations of a finite set of basis filters $h_1, \dots, h_M$:

$$h(\mathbf{R}_{-r}\mathbf{x}) = \sum_{m=1}^M c_m(r) h_m(\mathbf{x})$$

The interpolation functions $c_m(r)$ are trigonometric polynomials for steerable filters.

Circular harmonics. Filters of the form $h(\rho, \phi) = g(\rho)e^{in\phi}$ (polar coordinates) are steerable with $M = 1$:

$$h(\mathbf{R}_{-r}\mathbf{x}) = e^{inr} h(\mathbf{x})$$

A rotation by r multiplies the filter by a phase e^{inr} . The integer n is the *angular frequency* of the filter. Filters with $n = 0$ are radially symmetric; $n = 1$ gives oriented (edge-detector) filters; $n = 2$ gives corner detectors.

Group-equivariant CNN (G -CNN). Cohen & Welling (2016) extend convolution to group-valued feature maps. For an $SE(2)$ -equivariant network:

- Layer 1: convolve image $f : \mathbb{R}^2 \rightarrow \mathbb{R}$ with filters rotated at K angles: outputs a feature map $\Phi : SE(2) \rightarrow \mathbb{R}^C$.
- Subsequent layers: convolve feature maps over $SE(2)$ with $SE(2)$ -equivariant filters.
- Final layer: aggregate over rotations (making output invariant) via max or mean over the rotation axis.

EXAMPLE — $SE(2)$ -Equivariant CNN for Digit Recognition

MNIST rotated (random $[0^\circ, 360^\circ]$ rotation). Standard CNN: 78.1% test accuracy. $SE(2)$ -CNN ($K = 8$ rotations): 98.2% test accuracy.

The improvement comes from equivariance: the model sees each digit orientation as a transformed version of the same digit, rather than a new pattern. The effective data augmentation is $K = 8\times$ without any additional parameters.

For medical image analysis (pathology slides, microscopy), where objects can appear at any orientation, equivariant networks consistently outperform standard CNNs by 3–10% accuracy.

A *steerable filter* ψ_θ at orientation θ can be expressed as a fixed linear combination of basis filters $\{\psi_k\}$:

$$\psi_\theta = \sum_k c_k(\theta) \psi_k$$

This allows computing responses at all orientations from a finite filter bank. E(2)-equivariant networks (Cohen & Welling) generalise CNNs to be equivariant to all rotations, reflections, and translations of 2D images, achieving better parameter efficiency on tasks with rotational symmetry.

50.4 Spectral Graph Convolution and Graph Neural Networks

The Fourier analysis of §49.1 extends naturally to signals on graphs, providing the theoretical foundation for graph neural networks (GNNs) and spectral graph methods.

The graph Laplacian. For an undirected graph $\mathcal{G} = (V, E)$ with $|V| = n$ nodes and adjacency matrix $\mathbf{A}$, define:

- Degree matrix: $\mathbf{D} = \text{diag}(d_1, \dots, d_n)$ where $d_i = \sum_j \mathbf{A}_{ij}$.
- Combinatorial Laplacian: $\mathbf{L} = \mathbf{D} - \mathbf{A}$.
- Normalised Laplacian: $\tilde{\mathbf{L}} = \mathbf{D}^{-1/2} \mathbf{L} \mathbf{D}^{-1/2} = \mathbf{I} - \mathbf{D}^{-1/2} \mathbf{A} \mathbf{D}^{-1/2}$.

The normalised Laplacian has eigenvalues $0 = \lambda_1 \leq \lambda_2 \leq \dots \leq \lambda_n \leq 2$ with eigenvectors $\mathbf{U} = [\mathbf{u}_1, \dots, \mathbf{u}_n]$. The eigenvectors form the *graph Fourier basis*: low-frequency eigenvectors ($\lambda_i \approx 0$) vary slowly across the graph (similar values at connected nodes), while high-frequency eigenvectors ($\lambda_i \approx 2$) alternate in sign across edges.

Graph Fourier transform. For a graph signal $\mathbf{x} \in \mathbb{R}^n$ (one value per node):

$$\hat{\mathbf{x}} = \mathbf{U}^\top \mathbf{x} \quad (\text{graph Fourier transform})$$

$$\mathbf{x} = \mathbf{U} \hat{\mathbf{x}} \quad (\text{inverse graph Fourier transform})$$

Spectral graph convolution. Convolution on graphs is defined via the convolution theorem:

$$\mathbf{x} * \mathcal{G} \mathbf{g} = \mathbf{U}(\hat{\mathbf{x}} \odot \hat{\mathbf{g}}) = \mathbf{U} \text{diag}(\hat{\mathbf{g}}) \mathbf{U}^\top \mathbf{x}$$

A spectral filter $\hat{\mathbf{g}} = (g(\lambda_1), \dots, g(\lambda_n))$ applies a function g to each eigenvalue. The filter $g_\theta(\lambda) = \theta \lambda$ defines a linear spectral filter; $g_\theta(\lambda) = (1 - \theta \lambda)^k$ defines a k -hop polynomial filter.

The ChebNet approximation. Computing $\mathbf{U}$ and the full matrix product $\mathbf{U} \text{diag}(\hat{\mathbf{g}}) \mathbf{U}^\top$ costs $O(n^2)$. Defferrard et al. (2016) approximate $g(\lambda)$ with a Chebyshev polynomial of degree K :

$$g_\theta(\lambda) \approx \sum_{k=0}^K \theta_k T_k(\tilde{\lambda}), \quad \tilde{\lambda} = 2\lambda/\lambda_{\max} - 1$$

where $T_k(\cdot)$ is the Chebyshev polynomial of degree k . Since $T_k(\tilde{\mathbf{L}})\mathbf{x}$ can be computed recursively in $O(K|\mathcal{E}|)$ (using the sparsity of $\mathbf{L}$), the spectral convolution costs $O(K|\mathcal{E}|)$ — linear in graph size.

Graph Convolutional Network (GCN). Kipf & Welling (2017) simplify ChebNet to $K = 1$ with $\lambda_{\max} \approx 2$:

$$g_{\theta}(\lambda) \approx \theta_0 + \theta_1(1 - \lambda) \implies \mathbf{h} = \theta_0 \mathbf{x} + \theta_1 \mathbf{D}^{-1/2} \mathbf{A} \mathbf{D}^{-1/2} \mathbf{x}$$

Further setting $\theta_0 = \theta_1 = \theta$ and using the renormalisation trick $\tilde{\mathbf{A}} = \mathbf{A} + \mathbf{I}$:

$$\mathbf{H} = \sigma(\tilde{\mathbf{D}}^{-1/2} \tilde{\mathbf{A}} \tilde{\mathbf{D}}^{-1/2} \mathbf{X} \mathbf{W})$$

where $\tilde{\mathbf{D}} = \text{diag}(\tilde{\mathbf{A}}\mathbf{1})$. This is the GCN layer: each node aggregates the mean of its neighbours' features (weighted by degree), then applies a linear transform and activation. Cost: $O(|\mathcal{E}|d + nd')$ per layer.

GCN Layer (Kipf & Welling 2017)

For graph $\mathcal{G}$ with node features $\mathbf{X} \in \mathbb{R}^{n \times d}$ and weight matrix $\mathbf{W} \in \mathbb{R}^{d \times d'}$:

$$\mathbf{H} = \sigma(\hat{\mathbf{A}} \mathbf{X} \mathbf{W}), \quad \hat{\mathbf{A}} = \tilde{\mathbf{D}}^{-1/2} \tilde{\mathbf{A}} \tilde{\mathbf{D}}^{-1/2}$$

- $\hat{\mathbf{A}} \mathbf{X}$: computes the degree-normalised neighbourhood average for each node.
- $\hat{\mathbf{A}} \mathbf{X} \mathbf{W}$: projects to d' dimensions.
- $\sigma(\cdot)$: ReLU or similar non-linearity.

A L -layer GCN aggregates L -hop neighbourhood information, analogous to a CNN with receptive field of size L .

Over-smoothing. As $L \rightarrow \infty$, all node representations converge to the same vector (the top eigenvector of $\hat{\mathbf{A}}$), a phenomenon called *over-smoothing*. This limits GCN depth to $L = 2-4$ layers in practice. Remedies include residual connections (propagate input features to each layer) and PairNorm (normalise features to have unit pairwise distance).

EXAMPLE — GCN for Node Classification on Cora

Cora: citation network with 2708 papers (nodes), 5429 citation edges, 7 classes (machine learning topics).

Features: 1433-dim bag-of-words vector per paper. Labels: 140 training, 500 validation, 1000 test nodes.

Model	Test Accuracy
Logistic regression	74.5%
DeepWalk (no features)	67.2%
GCN (2 layers)	81.5%
GAT (graph attention, 2 layers)	83.0%

GCN achieves $\sim 7\%$ improvement over logistic regression by propagating features across the citation graph. The improvement is largest for classes with dense intra-class citations and sparse inter-class citations — consistent with the spectral theory: low-frequency eigenvectors (class structure) are amplified by GCN layers.

50.5 Equivariant Neural Networks and Group Theory

Convolutional networks are equivariant to translations by construction. The framework of *group-equivariant convolutions* (Cohen & Welling, 2016) generalises this to arbitrary symmetry groups, providing a principled way to build invariances into neural architectures.

Group equivariance. A function $f : \mathcal{X} \rightarrow \mathcal{Y}$ is *equivariant* to a group G acting on $\mathcal{X}$ and $\mathcal{Y}$ if:

$$f(g \cdot \mathbf{x}) = g \cdot f(\mathbf{x}) \quad \forall g \in G, \mathbf{x} \in \mathcal{X}$$

For a translation group $G = (\mathbb{R}^2, +)$, this is the familiar shift-equivariance of convolution: translating the input by $\square$ translates the feature maps by $\square$. *Invariance* is a special case: $f(g \cdot \mathbf{x}) = f(\mathbf{x})$ (pooling achieves invariance by averaging over the group orbit).

Group convolution. For a finite group G acting on $\mathbb{R}^2$ (e.g., rotation by 90° : $G = Z_4$), the group convolution is:

$$[f \star_G k](g) = \sum_{h \in G} \int_{\mathbb{R}^2} f(h^{-1} \cdot \mathbf{x}) k(g^{-1}h \cdot \mathbf{x}) d\mathbf{x}$$

This generalises standard convolution: for $G = (\mathbb{R}^2, +)$, it reduces to the standard convolution formula $[f \star k](\mathbf{t}) = \int f(\mathbf{x}) k(\mathbf{t} - \mathbf{x}) d\mathbf{x}$.

G-CNNs. Cohen & Welling (2016) define G-CNNs (group-equivariant CNNs) using group convolutions. For $G = p4$ (4-fold rotations + translations):

- Feature maps are defined on G rather than $\mathbb{R}^2$: $f : G \rightarrow \mathbb{R}^C$.
- The first layer lifts from $\mathbb{R}^2$ to G ; subsequent layers operate on $G \times G$.
- The final pooling averages over the group to achieve invariance.

G-CNNs achieve the same accuracy as standard CNNs with $4\times$ fewer parameters (for Z_4 symmetry), since the symmetry is built in rather than learned.

Steerable CNNs and irreducible representations. Cohen et al. (2016) generalise G-CNNs to continuous rotation groups $G = SO(2)$ or $SO(3)$ using the representation theory of G . A steerable feature is a vector that transforms according to a representation $\rho : G \rightarrow GL(V)$:

$$f(g^{-1}\mathbf{x}) = \rho(g)f(\mathbf{x})$$

Irreducible representations (irreps) of $SO(2)$ are complex exponentials $e^{im\phi}$ (labelled by integer m). A steerable CNN with irreps $m = 0, 1, 2$ represents features that are (respectively) rotationally invariant, equivariant to vector rotations, and equivariant to second-order tensor rotations.

Schur's Lemma for Network Design

Schur's lemma: For two irreducible representations ρ_1, ρ_2 of G , any equivariant linear map $L : V_1 \rightarrow V_2$ is zero if $\rho_1 \neq \rho_2$, and a scalar multiple of the identity if $\rho_1 = \rho_2$.

Network design implication: Equivariant layers can only mix feature channels of the same irrep type. This constrains the architecture: a $\rho_1 \rightarrow \rho_2$ convolutional layer is only non-trivially equivariant if $\rho_1 = \rho_2$ or ρ_2 appears in the decomposition of $\rho_1 \otimes \rho_{\text{filter}}$ (Clebsch-Gordan decomposition).

EXAMPLE — Equivariant Networks for Molecular Property Prediction

Predicting molecular properties (energy, forces) requires 3D $SO(3)$ -equivariance (atoms don't change property under rotation).

SchNet (Schütt et al., 2017): rotationally invariant (distances only, $m = 0$ scalars). MAE on QM9 energy: 0.63 kcal/mol.

DimeNet (Gasteiger et al., 2020): incorporates bond angles ($m = 0, 1$). MAE: 0.43

kcal/mol.

SE(3)-Transformer (Fuchs et al., 2020): full SE(3) equivariance (translations + rotations). MAE: 0.31 kcal/mol.

NequIP (Batzner et al., 2022): E(3)-equivariant network with $m = 0, 1, 2, 3$ irreps. MAE: 0.22 kcal/mol.

The progression shows: adding higher-order irreps systematically improves accuracy by capturing richer geometric interactions. Each irrep adds $O(2m+1)$ feature channels per “shell” at linear cost.

Adversarial Training and the Robust Optimisation View

Adversarial training is min-max optimisation. We analyse the Madry formulation, the trade-off between robustness and standard accuracy, and connections to distributionally robust optimisation.

51.1 The Adversarial Training Objective

Madry et al. (2017) proposed training on adversarial examples:

$$\min_{\square} \mathbb{E}_{(\mathbf{x}, y)} \left[\max_{\|\square\|_{\infty} \leq \epsilon} \mathcal{L}(f_{\square}(\mathbf{x} + \square), y) \right]$$

This is a *min-max* problem: the inner max finds the worst-case perturbation (using PGD), the outer min trains the model to be robust against it.

51.2 The Robustness-Accuracy Trade-off

Empirically, standard accuracy and adversarial robustness are at odds. Tsipras et al. (2019) showed that for certain data distributions, any classifier with high adversarial robustness must sacrifice some standard accuracy. Intuitively, robust features are coarser and less correlated with spurious features that are useful for standard classification.

Formal example. Consider a binary classification problem where $y \in \{+1, -1\}$ and:

$$\mathbf{x} = y \cdot \mathbf{v}_{\text{robust}} + \eta \cdot \mathbf{v}_{\text{non-robust}}, \quad \eta \sim \mathcal{N}(\mathbf{0}, \sigma^2 \mathbf{I})$$

A standard classifier can exploit $\mathbf{v}_{\text{non-robust}}$ (high signal-to-noise for clean data), but this feature is fragile to ℓ_{∞} perturbations. A robust classifier must rely on $\mathbf{v}_{\text{robust}}$, which has smaller signal and hence higher clean error.

51.3 Distributionally Robust Optimisation (DRO)

DRO generalises adversarial training to arbitrary distribution shifts:

$$\min_{\square} \sup_{Q: d(Q, P) \leq \rho} \mathbb{E}_Q[\mathcal{L}(f_{\square}(\mathbf{x}), y)]$$

With the Wasserstein- p distance as d , the inner supremum corresponds to a transportation problem. The Lagrangian relaxation gives:

$$\min_{\square} \mathbb{E}_P[\mathcal{L}] + \lambda \cdot W_p(Q, P)$$

which includes adversarial training ($p = \infty$, ℓ_{∞} ball) as a special case.

DRO with χ^2 divergence. An alternative to Wasserstein is the χ^2 divergence $d(Q, P) = \mathbb{E}_P[(dQ/dP - 1)^2]$. The DRO problem becomes:

$$\min_{\theta} \sup_{Q: \chi^2(Q \| P) \leq \rho} \mathbb{E}_Q[\mathcal{L}(\theta, z)]$$

Using the Lagrangian dual, the inner supremum has a closed-form solution:

$\sup Q = \text{“tilt” } P \text{ towards high-loss examples}$

Specifically, the optimal Q upweights examples with loss above a threshold τ :

$$\frac{dQ^*}{dP}(z) \propto \max\left(1 + \frac{\mathcal{L}(\theta, z) - \tau}{2\rho}, 0\right)$$

This is the CVaR (Conditional Value at Risk) reweighting: DRO with χ^2 divergence is equivalent to minimising the expected loss on the worst- α fraction of examples (tail risk), where α is determined by ρ .

EXAMPLE — DRO vs. ERM on Imbalanced Data

Binary classification, 90% class 0 and 10% class 1. Standard ERM minimises average loss, which can ignore class 1 (low frequency $\rightarrow$ low contribution to average loss).

DRO with $\chi^2, \rho = 0.1$: effectively upweights the harder class-1 examples.

Method	Class 0 Accuracy	Class 1 Accuracy
ERM	98.2%	61.4%
DRO ($\rho = 0.1$)	96.1%	84.7%

DRO sacrifices 2% on class 0 to gain 23% on class 1, improving worst-group accuracy. This is particularly valuable in fairness-aware ML, where “worst group” often corresponds to a minority demographic.

Group DRO. When the groups (subpopulations) are known a priori, Group DRO minimises the worst-group risk directly:

$$\min_{\theta} \max_{g \in \mathcal{G}} \mathbb{E}_{(x,y) \sim P_g} [\mathcal{L}(\theta, x, y)]$$

This is simpler to implement (just reweight by group) and empirically more effective than Wasserstein DRO when groups are available.

Summary of Volume IV and Bridges to Volume V

We consolidate the key mathematical results of this volume and preview the large-scale architecture mathematics of Volume V.

51.4 Connections between Robustness Frameworks

The adversarial robustness landscape has many frameworks — empirical (§43), certified (§44), differentially private (§45), and distribution-shift-robust (§46) — that are deeply interconnected.

Adversarial robustness and differential privacy. A striking connection: $(\varepsilon, 0)$ -DP implies adversarial robustness. If a classifier h satisfies pure DP (output invariant to single-point changes), then changing one pixel of an image (one data point) cannot change the prediction:

$$P[h(\mathbf{x} + \square) \neq h(\mathbf{x})] \leq 1 - e^{-\varepsilon}$$

for $\mathbf{x}'$ differing from $\mathbf{x}$ in one coordinate. This is a certified ℓ_0 robustness guarantee (one pixel can change). The connection goes deeper: randomised smoothing (§44) is a *local* DP mechanism (the Gaussian noise mechanism applied to the input), and the certified radius formula from §44.2 is exactly the hypothesis test corresponding to the Gaussian mechanism's privacy guarantee.

Randomised smoothing as a privacy mechanism. The smoothed classifier $g(\mathbf{x}) = \arg \max_c P[f(\mathbf{x} + \square) = c]$ with $\square \sim \mathcal{N}(0, \sigma^2 \mathbf{I})$ is:

- A private mechanism: adding Gaussian noise to the input provides (ε, δ) -DP for the input space, with $\varepsilon = \|\Delta_x\|_2 / \sigma$ where $\Delta_x = \mathbf{x} - \mathbf{x}'$ is the perturbation.
- The certified radius $R = \sigma \Phi^{-1}(p_A)$ is exactly the DP parameter: the maximum ℓ_2 perturbation that cannot be detected by the Neyman-Pearson test at significance level p_A .

Distributionally robust optimisation as adversarial training. DRO (§51.3) with Wasserstein- ∞ ball:

$$\sup_{Q: W_\infty(Q, P) \leq \varepsilon} \mathbb{E}_Q[\mathcal{L}(f, z)] = \mathbb{E}_P \left[\max_{\|z' - z\|_\infty \leq \varepsilon} \mathcal{L}(f, z') \right]$$

equals exactly the adversarial training objective of Madry et al. (2017). DRO with Wasserstein-2 ball corresponds to adversarial training with ℓ_2 perturbations; DRO with f -divergence balls corresponds to importance-weighted training with bounded density ratios.

Domain adaptation as robust optimisation. The Ben-David bound (§46.2) can be written as a DRO problem: the target risk $R_T(h)$ is the supremum of the source risk over all distributions in the $d_{\mathcal{H}}$ -divergence ball of radius $d_{\mathcal{H}}(P_S, P_T)/2$ around P_S :

$$R_T(h) \leq \sup_{Q: d_{\mathcal{H}}(Q, P_S) \leq d_{\mathcal{H}}(P_S, P_T)/2} R_Q(h) + \lambda^*$$

DANN (§46.3) minimises this DRO objective by learning domain-invariant features that make $d_{\mathcal{H}}$ small.

Unified view: robustness = worst-case over a ball. All robustness frameworks can be unified as:

$$\min_{\theta} \sup_{Q \in \mathcal{U}(P)} \mathbb{E}_Q[\mathcal{L}(f_\theta, z)]$$

where $\mathcal{U}(P)$ is an uncertainty set around the training distribution P :

Uncertainty set $\mathcal{U}(P)$	Robustness notion
ℓ_∞ ball in input space	Adversarial robustness (Madry)
Wasserstein-2 ball	Input metric robustness
f -divergence ball	Importance-weighted covariate shift
$\mathcal{H}$ -divergence ball	Domain adaptation (Ben-David)
(ε, δ) -DP neighbourhood	Differential privacy

This unification reveals that all of these frameworks are solving the same mathematical problem with different geometries on the space of distributions.

52.1 Core Results of Volume IV

Part	Key Mathematical Result
39 — Loss landscape	Saddle proliferation: probability of local minimum $\approx 2^{-d}$; flat minima $\Leftrightarrow$ better generalisation.
40 — Natural gradient	$\tilde{\nabla} \mathcal{L} = \mathbf{F}^{-1} \nabla \mathcal{L}$; K-FAC reduces inversion from $O(d^3)$ to two small per-layer inversions.
41 — PAC-Bayes	$R(Q) \leq \hat{R}(Q) + \sqrt{(\text{KL}(Q\ P) + \ln(n/\delta))/(2n)}$; non-vacuous bounds from compressibility.
42 — Information Bottleneck	$\min I(X; Z) - \beta I(Z; Y)$; layers compress irrelevant information.
43 — Adversarial attacks	FGSM: $\mathbf{x}' = \mathbf{x} + \epsilon \text{sign}(\nabla_{\mathbf{x}} \mathcal{L})$; PGD: iterated projected gradient.
44 — Certified robustness	Gaussian smoothing certificate: $R = \frac{\sigma}{2}(\Phi^{-1}(\underline{p}_A) - \Phi^{-1}(\bar{p}_B))$.
45 — Differential privacy	Gaussian mechanism: $\sigma \geq \Delta_2 \sqrt{2 \ln(1.25/\delta)}/\varepsilon$; DP-SGD privacy accounting.
46 — Domain adaptation	Ben-David: $\varepsilon_T \leq \varepsilon_S + \frac{1}{2}d_{\mathcal{H}+\lambda}$; DANN via gradient reversal.
47 — Neural ODEs	Adjoint method: $O(1)$ memory vs $O(T)$; continuous normalising flows via $\text{tr}(\partial f / \partial \mathbf{h})$.
48 — PMP and HJB	Adjoint ODE = costate equation; HJB PDE $-V_t = \min_u [r + (\nabla V)^\top f]$.
49 — Signal processing	Nyquist: $f_s > 2B$ for perfect reconstruction; blur-pooling prevents CNN aliasing.
50 — Equivariance	Convolution commutes with translation; steerable filters achieve $SE(2)$ equivariance.
51 — Adversarial training	Min-max: $\min_\theta \mathbb{E}[\max_\delta \mathcal{L}]$; robustness-accuracy trade-off; DRO as generalisation.

52.2 Mathematical Unification: Geometry, Information, and Robustness

The fourteen parts of Volume IV can be organised around three unifying mathematical themes: the *geometry of learning*, the *information theory of generalisation*, and the *robustness-optimality duality*.

Theme 1: Geometry of the parameter space. Parts 39–40 develop the intrinsic geometry of the loss surface and parameter space:

- The loss function defines a Riemannian hypersurface whose critical-point structure is governed by the Hessian spectrum (Part 39).
- Gradient descent ignores this geometry; natural gradient descent respects it by pre-conditioning with the Fisher metric (Part 40).
- The flat-minima conjecture connects local Hessian geometry to global generalisation.

The key insight is that the “right” geometry for learning is the Fisher-Rao geometry on the statistical manifold $\{p_\theta\}$, not the Euclidean geometry on parameter space $\mathbb{R}^d$.

Theme 2: Information-theoretic generalisation. Parts 41–42 frame generalisation as an information problem:

- PAC-Bayes bounds: the generalisation gap is bounded by $\sqrt{\text{KL}(Q\|P)/n}$, where $\text{KL}(Q\|P)$ measures the “information” the training algorithm extracts from the data (Part 41).
- Mutual information bounds: $\mathbb{E}[R(W) - \hat{R}(W)] \leq \sqrt{2I(W; S)/n}$, where $I(W; S)$ is the mutual information between learned weights and training data.
- Information bottleneck: good representations minimise $I(\mathbf{X}; \mathbf{Z})$ while preserving $I(\mathbf{Z}; Y)$ — the Pareto frontier of the compression-accuracy tradeoff (Part 42).

All three perspectives converge on the same conclusion: *compressibility implies generalisation*. Models that can be described compactly relative to a prior generalise well, regardless of their parameter count.

Theme 3: Robustness and adversarial mathematics. Parts 43–46, 51 develop the mathematics of robustness:

- Adversarial attacks (Parts 43–44): finding the worst-case perturbation is an optimisation problem; certifying robustness requires upper-bounding the worst-case loss.
- Differential privacy (Part 45): bounding information leakage is equivalent to bounding the sensitivity of the mechanism.
- Domain adaptation (Part 46): the Ben-David bound decomposes target risk into source risk + domain divergence + irreducible error.
- Adversarial training (Part 51): min-max optimisation bridges empirical risk minimisation and distributionally robust optimisation.

The connecting thread: duality. A recurring mathematical structure throughout the volume is Lagrangian duality:

Primal problem	Dual formulation	Connection
Natural gradient descent	Constrained optimisation on KL-ball	§40.2
PAC-Bayes generalisation	Donsker-Varadhan variational formula	§41.2
Information Bottleneck	Rate-distortion theory	§42.2
Adversarial training	Distributionally robust optimisation	§51.3
Gaussian mechanism (DP)	Hypothesis testing (Neyman-Pearson)	§44.2

Each of these connections reveals the same fundamental insight: the optimal solution to a min-max problem is determined by the geometry of the space of distributions, not by the geometry of the parameter space.

Looking ahead: Volume V. The themes of Volume IV scale up in Volume V to the mathematics of large language models and emergent capabilities:

- The geometry of attention (Parts 53–54) extends natural gradient theory to the manifold of attention patterns.
- Scaling laws (Part 58) connect information-theoretic compression (PAC-Bayes KL) to empirical power-law behaviour of loss vs. model size.
- Emergence (Part 59) is a phase transition in the information-theoretic generalisation landscape: below a critical model size, the IB curve is flat; above it, a sharp improvement enables in-context learning.

52.3 Key Theorems and Results in Volume IV

The following table summarises the major theoretical results of this volume for quick reference.

Result	Statement (informal)	Section
Saddle proliferation	Prob. of local min $\approx 2^{-d}$; critical points in high dim are almost all saddles.	§39.2
Flat minima / generalisation	Flatter minima (low $\lambda_{\max}$) generalise better; small-batch SGD finds flatter minima.	§39.3
Natural gradient (Amari 1998)	$\tilde{\nabla} \mathcal{L} = \mathbf{F}^{-1} \nabla \mathcal{L}$; invariant under reparametrisation; converges in $O(1)$ steps.	§40.1
K-FAC	$\mathbf{F}_l \approx \mathbf{A}_l \otimes \mathbf{G}_l$; inversion cost $O(n^3 + m^3)$ vs. $O((mn)^3)$ for full $\mathbf{F}$.	§40.3
PAC-Bayes bound (McAllester 1999)	$R(Q) \leq \hat{R}(Q) + \sqrt{(\text{KL}(Q\ P) + \ln(n/\delta))/(2n)}$ First non-vacuous NN bound (Dziugaite & Roy 2017).	§41.2
Info Bottleneck (Tishby 1999)	$\min I(X; Z) - \beta I(Z; Y)$; optimal encoder is a minimal sufficient statistic for Y .	§42.2
FGSM attack	$\mathbf{x}' = \mathbf{x} + \epsilon \text{sign}(\nabla_{\mathbf{x}} \mathcal{L})$; one-step ℓ_{∞} -maximiser of linearised loss.	§43.2
PGD attack (Madry 2017)	Iterated projected gradient; gold standard for evaluating adversarial robustness.	§43.3
Randomised smoothing (Cohen 2019)	Certified ℓ_2 radius $R = \frac{\sigma}{2}(\Phi^{-1}(\underline{p}_A) - \Phi^{-1}(\bar{p}_B))$ by Neyman-Pearson halfspace geometry.	§44.2
Gaussian mechanism (Dwork 2006)	$\sigma \geq \Delta_2 \sqrt{2 \ln(1.25/\delta)}/\epsilon$ achieves (ϵ, δ) -DP; DP-SGD clips + noises gradients.	§45.2
Ben-David bound	$\varepsilon_T \leq \varepsilon_S + \frac{1}{2} d_{\mathcal{H}}(\mathcal{D}_S, \mathcal{D}_T) + \lambda^*$ DANN minimises $d_{\mathcal{H}}$ via gradient reversal.	§46.2
Adjoint method (Pontryagin 1956)	Backward ODE for $\mathbf{a}(t)$ computes gradients in $O(1)$ memory vs. $O(T)$ for BPTT.	§47.2
Pontryagin MP	Optimal control requires $H(\mathbf{x}^*, \mathbf{u}^*, \mathbf{p}^*) = \text{global max over } \mathbf{u}$; PMP $\supset$ Euler-Lagrange.	§48.2
Nyquist theorem	Exact reconstruction iff $f_s > 2B$; aliasing in CNNs cured by blur-pooling.	§49.2
Equivariant conv	Conv commutes with translation; steerability extends this to $SE(2)$, $SE(3)$ symmetries.	§50.1–50.3

52.4 Bridges to Volume V

Volume V escalates from single-model mathematics to the mathematics of *scale*. The concepts developed in Volume IV provide the necessary scaffolding for understanding large-scale architectures.

From optimisation geometry to attention. The natural gradient framework (Part 40) extends directly to the Transformer attention mechanism. The attention matrix $\mathbf{A} = \text{softmax}(\mathbf{QK}^T / \sqrt{d})$ defines a Riemannian metric on sequence space, and the “information geometry of attention” (Volume V, Parts 53–54) develops the Fisher-Rao metric for softmax distributions. The key insight: efficient attention variants (Flash-Attention, GQA, sliding window) are approxima-

tions to the natural gradient step that preserve the metric structure while reducing $O(n^2)$ to $O(n \log n)$.

From neural ODEs to state space models. The continuous-depth perspective of Part 47 provides the theoretical basis for state space models (SSMs). The S4 model (Gu et al., 2022) defines the sequence transformation:

$$\dot{\mathbf{h}}(t) = \mathbf{A}\mathbf{h}(t) + \mathbf{B}x(t), \quad y(t) = \mathbf{C}\mathbf{h}(t)$$

as a continuous-time linear dynamical system. The structured matrix $\mathbf{A}$ (HiPPO matrix) is designed so that the ODE's solution captures long-range dependencies efficiently. The adjoint method (§47.2) applies directly to compute gradients through the ODE solver. Mamba (Gu & Dao, 2023) makes the matrices $\mathbf{A}$, $\mathbf{B}$, $\mathbf{C}$ input-dependent, turning the linear ODE into a selective state space model.

From PAC-Bayes to scaling laws. The PAC-Bayes bound (Part 41) provides a theoretical lens for scaling laws. For a neural network with parameter count N trained on D tokens, the Chinchilla scaling laws are:

$$L(N, D) \approx \frac{A}{N^\alpha} + \frac{B}{D^\beta} + E$$

where E is the irreducible entropy of language. The two terms correspond to the PAC-Bayes bound: A/N^α is the *model capacity term* (larger models have lower KL from prior to posterior) and B/D^β is the *data efficiency term* (more data reduces the generalisation gap). The exponents $\alpha \approx 0.34$ and $\beta \approx 0.28$ (Hoffmann et al., 2022) are universal across architectures, suggesting they reflect fundamental properties of the information geometry of language.

From adversarial training to RLHF. The min-max formulation of adversarial training (Part 51) is directly mirrored in RLHF (Reinforcement Learning from Human Feedback, Volume V, Part 62). In RLHF:

- The *policy* π_θ (language model) plays the role of the defender.
- The *reward model* r_ϕ plays the role of the attacker (finding responses that maximise reward).
- The KL constraint $\text{KL}(\pi_\theta \| \pi_{\text{ref}}) \leq \epsilon$ is the DRO uncertainty set (§51.3).

The PPO-clip objective (§48.4) is the standard RLHF training algorithm, with the policy gradient providing the gradient signal and the KL penalty ensuring the policy stays close to the reference (SFT) model.

From information bottleneck to mechanistic interpretability. The IB framework (Part 42) provides a theoretical foundation for understanding what information each layer of a transformer computes. Mechanistic interpretability (Volume V, Part 64) studies specific circuits (attention heads + MLP weights) that implement particular computations. The superposition hypothesis (Elhage et al., 2022) states that networks with d neurons can store $n \gg d$ features by representing them as nearly-orthogonal vectors in $\mathbb{R}^d$, with interference bounded by $1/\sqrt{d}$. This is a direct consequence of the IB: the network maximises $I(\text{features}; \mathbf{h})$ by using nearly all the $\log d$ bits available in a d -dimensional representation, even when storing n binary features.

Volume IV concept	Volume V extension	Key connection
Natural gradient (§40)	Attention geometry	Fisher metric on softmax
Neural ODEs (§47)	State space models	Continuous-time SSM
PAC-Bayes (§41)	Scaling laws	Capacity vs. data tradeoff
Adversarial training (§51)	RLHF / PPO	Min-max policy optimisation
Information bottleneck (§42)	Mechanistic interp.	Superposition hypothesis
Differential privacy (§45)	Federated LLMs	DP-SGD at scale
Domain adaptation (§46)	Cross-lingual transfer	Distribution shift in NLP

Volume V thus builds directly on the mathematical foundations of Volume IV, scaling the same principles (geometry, information, robustness) to the regime of billion-parameter models, trillion-token datasets, and emergent capabilities.

Appendici Tecniche

Dimostrazioni Differite

A.1 Bound PAC-Bayes di McAllester

Teorema (McAllester 1999, Catoni 2007)

Sia P una distribuzione a priori su $\mathcal{H}$, fissata prima di vedere i dati. Per qualunque distribuzione a posteriori Q su $\mathcal{H}$ e qualunque $\delta \in (0, 1)$, con probabilità almeno $1 - \delta$ sul campione $S \sim \mathcal{D}^n$:

$$R(Q) \leq \hat{R}(Q) + \sqrt{\frac{\text{KL}(Q\|P) + \ln(n/\delta)}{2(n-1)}}.$$

Dimostrazione (schema via cambio di misura). Per qualunque funzione $\phi : \mathcal{H} \times \mathcal{Z}^n \rightarrow \mathbb{R}$ si ha l'identità di cambio di misura:

$$\ln \mathbb{E}_{h \sim P} [e^{\phi(h, S)}] = \sup_Q \{ \mathbb{E}_{h \sim Q} [\phi(h, S)] - \text{KL}(Q\|P) \}.$$

Applicando questa identità con $\phi(h, S) = n(R(h) - \hat{R}(h))$ e usando Hoeffding per controllare $\mathbb{E}_S[e^\phi]$, si ottiene che con probabilità $1 - \delta$ su S :

$$\mathbb{E}_{h \sim Q} [R(h) - \hat{R}(h)] \leq \sqrt{\frac{\text{KL}(Q\|P) + \ln(1/\delta)}{2n}}.$$

La versione con n/δ invece di $1/\delta$ segue dall'union bound su tutti i livelli di confidenza. $\square$

Il bound diventa non-vacuo per reti neurali solo quando Q è una distribuzione gaussiana sull'intorno di un minimo flat (bassa Hessiana). Dziugaite & Roy (2017) ottengono il primo bound non-vacuo ($\approx 26\%$ su MNIST) ottimizzando Q con SGD stocastico vincolato.

A.2 Teorema di certificazione del Randomised Smoothing

Teorema (Cohen, Rosenfeld & Kolter 2019)

Sia $f : \mathbb{R}^d \rightarrow \mathcal{Y}$ un classificatore base e $g(\mathbf{x}) = \arg \max_c \Pr_{\delta \sim \mathcal{N}(0, \sigma^2 \mathbf{I})}[f(\mathbf{x} + \delta) = c]$ il classificatore smoothed. Supponiamo $g(\mathbf{x}) = c_A$ con

$$\underline{p}_A \triangleq \Pr[f(\mathbf{x} + \delta) = c_A] \geq \bar{p}_B \triangleq \max_{c \neq c_A} \Pr[f(\mathbf{x} + \delta) = c].$$

Allora $g(\mathbf{x} + \square) = c_A$ per ogni perturbazione $\|\square\|_2 < R$, dove:

$$R = \frac{\sigma}{2} \left(\Phi^{-1}(\underline{p}_A) - \Phi^{-1}(\bar{p}_B) \right).$$

Dimostrazione. La prova usa il lemma di Neyman-Pearson: tra tutte le regioni di $\mathbb{R}^d$ con probabilità gaussiana $\geq \underline{p}_A$, quella che minimizza la probabilità dopo traslazione di $\square$ è il semispazio $\{\mathbf{z} : \mathbf{u}^\top \mathbf{z} \geq t\}$ per qualche $\mathbf{u}$ e soglia t . Pertanto la probabilità della classe avversaria soddisfa:

$$\Pr[f(\mathbf{x} + \square + \delta) = c_A] \geq \Phi \left(\Phi^{-1}(\underline{p}_A) - \frac{\|\square\|_2}{\sigma} \right).$$

Imponendo che questa probabilità superi $\bar{p}_B$ si ottiene la formula del raggio R . $\square$

A.3 Meccanismo Gaussiano e Privacy Differenziale

Definizione (ϵ, δ) -DP

Un meccanismo randomizzato $\mathcal{M} : \mathcal{X}^n \rightarrow \mathcal{R}$ soddisfa (ϵ, δ) -DP se per ogni coppia di dataset adiacenti $D \simeq D'$ e ogni evento $S \subseteq \mathcal{R}$:

$$\Pr[\mathcal{M}(D) \in S] \leq e^\epsilon \Pr[\mathcal{M}(D') \in S] + \delta.$$

Dimostrazione del Meccanismo Gaussiano. La sensibilità ℓ_2 di una funzione f è $\Delta_2 = \max_{D \simeq D'} \|f(D) - f(D')\|_2$. Il meccanismo $\mathcal{M}(D) = f(D) + \mathcal{N}(0, \sigma^2 \mathbf{I})$ soddisfa (ϵ, δ) -DP se

$$\sigma \geq \frac{\Delta_2 \sqrt{2 \ln(1.25/\delta)}}{\epsilon}.$$

Proof sketch: il rapporto di verosimiglianza tra $\mathcal{N}(f(D), \sigma^2 \mathbf{I})$ e $\mathcal{N}(f(D'), \sigma^2 \mathbf{I})$ è distribuito come $\exp(\mathcal{N}(\mu, \mu^2))$ con $\mu = \|f(D) - f(D')\|_2 / \sigma \leq \Delta_2 / \sigma$. Applicando la coda della gaussiana si ottiene la formula per σ . $\square$

A.4 Metodo Aggiunto per Neural ODE

Sia $\mathbf{h}(t)$ la soluzione dell'ODE $\dot{\mathbf{h}} = f_\theta(\mathbf{h}, t)$ con $\mathbf{h}(t_0) = \mathbf{h}_0$. Definire la variabile aggiunta $\mathbf{a}(t) = \partial \mathcal{L} / \partial \mathbf{h}(t)$. Essa soddisfa l'ODE retrograde:

$$\dot{\mathbf{a}}(t) = -\mathbf{a}(t)^\top \frac{\partial f_\theta}{\partial \mathbf{h}}, \quad \mathbf{a}(T) = \frac{\partial \mathcal{L}}{\partial \mathbf{h}(T)}.$$

Il gradiente rispetto ai parametri si ottiene integrando:

$$\frac{\partial \mathcal{L}}{\partial \theta} = - \int_T^{t_0} \mathbf{a}(t)^\top \frac{\partial f_\theta}{\partial \theta} dt.$$

Vantaggio di memoria. Invece di memorizzare le attivazioni a ogni passo (costo $O(T)$ come BPTT), è sufficiente risolvere l'ODE aggiunta integrando a ritroso con un ODE solver adattivo. Il costo di memoria è $O(1)$ rispetto alla profondità. $\square$

A.5 Bound di Ben-David per Domain Adaptation

Teorema (Ben-David et al. 2010)

Sia $\varepsilon_S(h)$ il rischio su dominio sorgente, $\varepsilon_T(h)$ su dominio target, $\lambda^* = \min_h [\varepsilon_S(h) + \varepsilon_T(h)]$ il rischio ideale congiunto, e $d_{\mathcal{H}(\mathcal{D}_S, \mathcal{D}_T)}$ la distanza $\mathcal{H}$ -divergenza. Allora:

$$\varepsilon_T(h) \leq \varepsilon_S(h) + \frac{1}{2} d_{\mathcal{H}(\mathcal{D}_S, \mathcal{D}_T) + \lambda^*}.$$

Proof sketch: si scrive $\varepsilon_T(h) - \varepsilon_S(h) = \mathbb{E}_{\mathcal{D}_T}[\ell(h)] - \mathbb{E}_{\mathcal{D}_S}[\ell(h)]$. Questo gap è al massimo $d_{\mathcal{H}/2}$ per definizione di $\mathcal{H}$ -divergenza. Aggiungendo e sottraendo l'ipotesi ottimale h^* si ottiene il termine λ^* . $\square$

Ricettario di Ottimizzazione Avanzata e Robustezza

B.1 Tassonomia degli attacchi avversariali

Attacco	Norma	Passi	Conoscenza	Tipico ε
FGSM	ℓ_∞	1	White-box	8/255
PGD- k	ℓ_∞	k	White-box	8/255, $k = 40$
C&W	ℓ_2	iterativo	White-box	min ℓ_2
AutoAttack	mista	adatt.	White-box	standard benchmark
NES / SPSA	ℓ_∞	iterativo	Black-box (score)	8/255
Square Attack	ℓ_∞	iterativo	Black-box (score)	8/255
Boundary Attack	ℓ_2	decision	Black-box (decision)	min ℓ_2

B.2 Algoritmi di ottimizzazione del secondo ordine

Metodo	Update	Costo/step
Newton	$\theta \leftarrow \theta - \mathbf{H}^{-1} \nabla \mathcal{L}$	$O(P^3)$
L-BFGS	approx. $\mathbf{H}^{-1}$ con m vettori	$O(mP)$
Natural gradient	$\theta \leftarrow \theta - \mathbf{F}^{-1} \nabla \mathcal{L}$	$O(P^3)$
K-FAC	$\mathbf{F}_l \approx \mathbf{A}_l \otimes \mathbf{G}_l$	$O(n_l^3 + n_{l-1}^3)$
Shampoo	fattori di Kronecker per layer	$O(n^3)$ per blocco

K-FAC in dettaglio. Per uno strato lineare $\mathbf{z} = \mathbf{W}\mathbf{a}$, il blocco della FIM è:

$$\mathbf{F}_l = \mathbb{E}[\mathbf{a}\mathbf{a}^\top \otimes \mathbf{g}\mathbf{g}^\top] \approx \mathbb{E}[\mathbf{a}\mathbf{a}^\top] \otimes \mathbb{E}[\mathbf{g}\mathbf{g}^\top] = \mathbf{A}_l \otimes \mathbf{G}_l,$$

dove $\mathbf{g} = \partial \mathcal{L} / \partial \mathbf{z}$ è il gradiente delle pre-attivazioni. L'inversa si fattorizza: $(\mathbf{A}_l \otimes \mathbf{G}_l)^{-1} = \mathbf{A}_l^{-1} \otimes \mathbf{G}_l^{-1}$. Il passo K-FAC è:

$$\Delta \mathbf{W}_l = \mathbf{G}_l^{-1} \nabla_{\mathbf{W}_l} \mathcal{L} \mathbf{A}_l^{-1}.$$

B.3 DP-SGD: schema algoritmico

1. Per ogni mini-batch $\mathcal{B}_t$ di taglia q :

- (a) Calcolare il gradiente per campione: $\mathbf{g}_i = \nabla_{\theta} \ell(\theta; x_i)$
- (b) Clipping: $\tilde{\mathbf{g}}_i = \mathbf{g}_i / \max(1, \|\mathbf{g}_i\|_2 / C)$ (sensibilità $\leq C$)
- (c) Rumore: $\hat{\mathbf{g}} = \frac{1}{|\mathcal{B}_t|} (\sum_i \tilde{\mathbf{g}}_i + \mathcal{N}(0, \sigma^2 C^2 \mathbf{I}))$
- (d) Update: $\theta_{t+1} = \theta_t - \alpha \hat{\mathbf{g}}$

2. Tracciamento del budget: usare il Moments Accountant (o RDP) per sommare il consumo di privacy a ogni passo.

Privacy amplification. Con campionamento Poisson di probabilità $q = |\mathcal{B}|/n$, il meccanismo amplificato soddisfa $(\tilde{\epsilon}, \tilde{\delta})$ -DP con $\tilde{\epsilon} \approx q\epsilon$ per ϵ piccolo.

B.4 Formulazioni equivalenti della DRO

La Distributionally Robust Optimisation (DRO) con set di incertezza $\mathcal{U}$:

$$\min_{\theta} \max_{Q \in \mathcal{U}(\mathcal{D})} \mathbb{E}_{(x,y) \sim Q} [\ell(\theta; x, y)]$$

ammette formulazioni duali secondo la scelta di $\mathcal{U}$:

Set $\mathcal{U}$	Formulazione duale	Connessione
$W_2(\mathcal{D}, Q) \leq \rho$	$\min_{\theta} \mathbb{E}_{\mathcal{D}[\ell] + \rho \sqrt{\mathbb{E}[\ \nabla_x \ell\ ^2]}}$	Smoothness
$\text{KL}(Q \ \mathcal{D}) \leq \rho$	$\min_{\theta} \frac{1}{\lambda} \ln \mathbb{E}_{\mathcal{D}[e^{\lambda \ell}]}$	CVaR / Tilted ERM
$\chi^2(Q \ \mathcal{D}) \leq \rho$	$\min_{\theta} \mathbb{E}[\ell] + \rho \sqrt{\text{Var}[\ell]}$	Variance reg.
ℓ_{∞} -ball	max-loss su attacchi ℓ_{∞}	Adversarial training

Complessità e Tradeoff Computazionali

Operazione	Tempo	Memoria	Note
<i>Ottimizzazione del secondo ordine</i>			
Newton (full)	$O(P^3)$	$O(P^2)$	Inattuabile per DNN
L-BFGS (m vettori)	$O(mP)$	$O(mP)$	$m = 10\text{--}20$ tipico
K-FAC (per layer)	$O(n_l^3 + n_{l-1}^3)$	$O(n_l^2 + n_{l-1}^2)$	vs. $O(n_l^3 n_{l-1}^3)$ full
Shampoo	$O(\sum_l \min(n_l, n_{l-1})^3)$	$O(\sum_l n_l^2)$	Distribuito su TPU
<i>Robustezza avversariale</i>			
FGSM	$O(P)$	$O(P)$	1 backward pass
PGD- k	$O(kP)$	$O(P)$	k backward passes
Randomised Smoothing	$O(nP)$	$O(P)$	n forward passes
IBP (Interval Bound)	$O(LP)$	$O(LP)$	$L =$ profondità
<i>Privacy differenziale</i>			
DP-SGD (per-sample grad)	$O(BP)$	$O(BP)$	$B =$ batch size
DP-SGD (ghostclip)	$O(BP)$	$O(P)$	Senza gradi per campione
Moments Accountant	$O(T \log T)$	$O(T)$	$T =$ n. step
<i>Neural ODE</i>			
Forward (Euler, T step)	$O(TP)$	$O(P)$	Adjoint: $O(1)$ memoria
Adjoint backward	$O(TP)$	$O(P)$	vs. BPTT $O(TP)$
Normalising flow (CNF)	$O(Td^3)$	$O(d^2)$	Stima della traccia
<i>Segnale e spettro</i>			
DFT / FFT	$O(n \log n)$	$O(n)$	Cooley-Tukey
Conv spettrale	$O(n \log n + k)$	$O(n)$	vs. $O(nk)$ spaziale
Filtro blur-pooling	$O(nk^2)$	$O(n)$	Prevenzione aliasing

C.1 Tradeoff accuratezza-robustezza

Il teorema di Tsipras et al. (2019) mostra che il tradeoff tra accuratezza standard acc_{std} e accuratezza robusta acc_{rob} è fondamentale per distribuzioni con separazione lineare imperfetta:

$$\text{acc}_{\text{std}} + \text{acc}_{\text{rob}} \leq 1 + \Pr_{(x,y) \sim \mathcal{D}} [\exists x' : \|x' - x\|_{\infty} \leq \varepsilon, f_{\theta}(x') = y],$$

dove il termine di destra è la probabilità che l' ε -intorno di x contenga campioni di classe y . Per distribuzioni sovrappesse, questo può essere significativamente < 1 , rendendo il tradeoff inevitabile anche per classificatori ottimali.

C.2 Complessità campionaria della certificazione

Per stimare $p_A = \Pr[f(\mathbf{x} + \delta) = c_A]$ con errore $\pm\alpha$ e confidenza $1 - \beta$, sono necessari:

$$n_{\text{cert}} = \frac{1}{4\alpha^2} \ln \frac{2}{\beta}$$

campioni gaussiani. Per $\alpha = 0.01$, $\beta = 0.001$: $n_{\text{cert}} \approx 8\,300$ forward pass per un singolo punto di test, il principale collo di bottiglia della randomised smoothing.

Glossario dei Simboli — Volume IV

Simbolo	Significato
<i>Ottimizzazione e loss landscape</i>	
$\mathbf{H} = \nabla^2 \mathcal{L}$	Matrice Hessiana della loss
$\lambda_{\max}(\mathbf{H})$	Massimo autovalore dell'Hessiana (sharpness)
$\mathbf{F}$	Matrice di informazione di Fisher
$\tilde{\nabla} \mathcal{L} = \mathbf{F}^{-1} \nabla \mathcal{L}$	Gradiente naturale
$\mathbf{A}_l, \mathbf{G}_l$	Fattori di Kronecker K-FAC (attivazioni, gradienti)
L	Costante di Lipschitz del gradiente
μ	Costante di forte convessità
$\kappa = L/\mu$	Numero di condizionamento
<i>Generalizzazione e teoria</i>	
$R(h), \hat{R}(h)$	Rischio vero, rischio empirico
P, Q	Prior, posterior (PAC-Bayes)
$\text{KL}(Q\ P)$	Divergenza di Kullback-Leibler
$\mathfrak{R}_n(\mathcal{H})$	Complessità di Rademacher
$\text{VC}(\mathcal{H})$	Dimensione VC della classe $\mathcal{H}$
$I(X; Z)$	Informazione mutua
β	Parametro di trade-off Information Bottleneck
<i>Robustezza avversariale</i>	
$\mathbf{x}' = \mathbf{x} + \square$	Esempio avversariale
ε	Raggio della perturbazione (ℓ_p -ball)
$\Delta \mathcal{L} = \mathcal{L}(\mathbf{x} + \square) - \mathcal{L}(\mathbf{x})$	Aumento della loss
$\underline{p}_A, \bar{p}_B$	Stime probabilità classe A e B (smoothing)
R	Raggio certificato (ℓ_2)
σ	Deviazione standard del rumore gaussiano
<i>Privacy differenziale</i>	
(ε, δ) -DP	Budget di privacy
$\Delta_2 f$	Sensibilità ℓ_2 della funzione f
C	Soglia di clipping (DP-SGD)
$q = B/n$	Probabilità di campionamento
σ_{DP}	Deviazione standard del rumore DP
<i>Domain adaptation</i>	
$\mathcal{D}_S, \mathcal{D}_T$	Distribuzione sorgente, target
$d_{\mathcal{H}(\mathcal{D}_S, \mathcal{D}_T)}$	$\mathcal{H}$ -divergenza
λ^*	Rischio ideale congiunto
$\mathcal{R}(\cdot)$	Funzione di rivolta (gradient reversal)
<i>Neural ODE e controllo</i>	
$\mathbf{h}(t)$	Stato continuo nel tempo
$\mathbf{a}(t) = \partial \mathcal{L} / \partial \mathbf{h}(t)$	Variabile aggiunta
$H(\mathbf{x}, \mathbf{u}, \mathbf{p})$	Hamiltoniana (Pontryagin)
$\mathbf{p}(t)$	Stato costato (variabile duale)
$V(\mathbf{x}, t)$	Funzione di valore (HJB)
<i>Teoria spettrale</i>	
f_s	Frequenza di campionamento (Nyquist)
B	Banda del segnale
$\hat{f}(\xi)$	Trasformata di Fourier di f
$*$	Convoluzione

Bibliografia

Loss landscape e geometria dell'ottimizzazione

1. Y. N. Dauphin, R. Pascanu, C. Gulcehre, K. Cho, S. Ganguli, and Y. Bengio, "Identifying and attacking the saddle point problem in high-dimensional non-convex optimization," in *Proc. NeurIPS*, 2014.
2. H. Li, Z. Xu, G. Taylor, C. Studer, and T. Goldstein, "Visualizing the loss landscape of neural nets," in *Proc. NeurIPS*, 2018.
3. L. Dinh, R. Pascanu, S. Bengio, and Y. Bengio, "Sharp minima can generalize for deep nets," in *Proc. ICML*, pp. 1019–1028, 2017.
4. P. Foret, A. Kleiner, H. Mobahi, and B. Neyshabur, "Sharpness-aware minimization for efficiently improving generalization (SAM)," in *Proc. ICLR*, 2021.
5. G. K. Cohen, E. Gal, S. Mannor, D. Soudry, and Y. Kapon, "Gradient descent on neural networks typically occurs at the edge of stability," in *Proc. ICLR*, 2022.

Ottimizzazione del secondo ordine e gradiente naturale

6. S. Amari, "Natural gradient works efficiently in learning," *Neural Computation*, vol. 10, no. 2, pp. 251–276, 1998.
7. J. Martens and R. Grosse, "Optimizing neural networks with Kronecker-factored approximate curvature (K-FAC)," in *Proc. ICML*, pp. 2408–2417, 2015.
8. D. C. Liu and J. Nocedal, "On the limited memory BFGS method for large scale optimization," *Mathematical Programming*, vol. 45, pp. 503–528, 1989.
9. V. Vyas, A. Morningstar, N. Garg, and V. Sharan, "Shampoo: Preconditioned stochastic tensor optimization," in *Proc. ICML*, 2018.

Teoria della generalizzazione

10. D. A. McAllester, "PAC-Bayesian model averaging," in *Proc. COLT*, pp. 164–170, 1999.
11. G. K. Dziugaite and D. M. Roy, "Computing nonvacuous generalization bounds for deep (stochastic) neural networks with many parameters," in *Proc. UAI*, 2017.
12. P. L. Bartlett and S. Mendelson, "Rademacher and Gaussian complexities: Risk bounds and structural results," *J. Machine Learning Research*, vol. 3, pp. 463–482, 2002.
13. M. Belkin, D. Hsu, S. Ma, and S. Mandal, "Reconciling modern machine learning practice and the classical bias-variance trade-off," *PNAS*, vol. 116, no. 32, pp. 15849–15854, 2019.
14. N. Shalev-Shwartz and S. Ben-David, *Understanding Machine Learning: From Theory to Algorithms*. Cambridge University Press, 2014.

Information Bottleneck

15. N. Tishby, F. C. Pereira, and W. Bialek, "The information bottleneck method," in *Proc. Allerton*, 1999.
16. R. Shwartz-Ziv and N. Tishby, "Opening the black box of deep neural networks via information," *arXiv preprint arXiv:1703.00810*, 2017.
17. A. Kolchinsky and B. D. Tracey, "Nonlinear information bottleneck," *Entropy*, vol. 21, no. 12, p. 1181, 2019.
18. M. I. Belghazi *et al.*, "Mutual information neural estimation (MINE)," in *Proc. ICML*, 2018.

Robustezza avversariale: attacchi

19. I. J. Goodfellow, J. Shlens, and C. Szegedy, "Explaining and harnessing adversarial examples (FGSM)," in *Proc. ICLR*, 2015.
20. A. Madry, A. Makelov, L. Schmidt, D. Tsipras, and A. Vladu, "Towards deep learning models resistant to adversarial attacks (PGD)," in *Proc. ICLR*, 2018.
21. N. Carlini and D. Wagner, "Towards evaluating the robustness of neural networks (C&W)," in *Proc. IEEE S&P*, pp. 39–57, 2017.
22. F. Croce and M. Hein, "Reliable evaluation of adversarial robustness with an ensemble of diverse parameter-free attacks (AutoAttack)," in *Proc. ICML*, 2020.

Robustezza certificata

23. J. M. Cohen, E. Rosenfeld, and Z. Kolter, "Certified adversarial robustness via randomized smoothing," in *Proc. ICML*, pp. 1310–1320, 2019.
24. G. Salman *et al.*, "Provably robust deep learning via adversarially trained smoothed classifiers (SmoothAdv)," in *Proc. NeurIPS*, 2019.
25. S. Gowal *et al.*, "Scalable verified neural network training via tight over-approximation (IBP)," *arXiv preprint arXiv:1810.12715*, 2018.
26. M. N. Müller *et al.*, "Prima: General and scalable model verification via property-driven abstract interpretation," in *Proc. NeurIPS*, 2022.

Privacy differenziale e federated learning

27. C. Dwork, F. McSherry, K. Nissim, and A. Smith, "Calibrating noise to sensitivity in private data analysis," in *Proc. TCC*, pp. 265–284, 2006.
28. M. Abadi *et al.*, "Deep learning with differential privacy (DP-SGD)," in *Proc. CCS*, pp. 308–318, 2016.
29. I. Mironov, "Rényi differential privacy of the Gaussian mechanism," in *Proc. CSF*, pp. 263–275, 2017.
30. H. B. McMahan *et al.*, "Communication-efficient learning of deep networks from decentralized data (FedAvg)," in *Proc. AISTATS*, 2017.

31. V. Smith, C.-K. Chiang, M. Sanjabi, and A. S. Talwalkar, "Federated multi-task learning," in *Proc. NeurIPS*, 2017.

Domain adaptation e distributional robustness

32. S. Ben-David, J. Blitzer, K. Crammer, A. Kulesza, F. Pereira, and J. W. Vaughan, "A theory of learning from different domains," *Machine Learning*, vol. 79, nos. 1–2, pp. 151–175, 2010.
33. Y. Ganin *et al.*, "Domain-adversarial training of neural networks (DANN)," *J. Machine Learning Research*, vol. 17, no. 59, pp. 1–35, 2016.
34. D. Shen *et al.*, "Wasserstein distance guided representation learning for domain adaptation," in *Proc. AAAI*, 2018.
35. J. Duchi and H. Namkoong, "Learning models with uniform performance via distributionally robust optimization," *Annals of Statistics*, vol. 49, no. 3, pp. 1378–1406, 2021.

Neural ODE e controllo ottimale

36. R. T. Q. Chen, Y. Rubanova, J. Bettencourt, and D. Duvenaud, "Neural ordinary differential equations," in *Proc. NeurIPS*, 2018.
37. L. S. Pontryagin, V. Boltyanskii, R. Gramkrelidze, and E. Mishchenko, *The Mathematical Theory of Optimal Processes*. Wiley-Interscience, 1962.
38. L. Grüne and J. Pannek, *Nonlinear Model Predictive Control: Theory and Algorithms*, 2nd ed. Springer, 2017.
39. W. Grathwohl, R. T. Q. Chen, J. Bettencourt, I. Sutskever, and D. Duvenaud, "FFJORD: Free-form continuous dynamics for scalable reversible generative models," in *Proc. ICLR*, 2019.
40. M. Raissi, P. Perdikaris, and G. E. Karniadakis, "Physics-informed neural networks (PINNs)," *J. Computational Physics*, vol. 378, pp. 686–707, 2019.

Equivarianza, spettro e segnali

41. R. Zhang, "Making convolutional networks shift-invariant again (blur-pooling)," in *Proc. ICML*, 2019.
42. T. S. Cohen and M. Welling, "Steerable CNNs," in *Proc. ICLR*, 2017.
43. M. Weiler and G. Cesa, "General $E(2)$ -equivariant steerable CNNs," in *Proc. NeurIPS*, 2019.
44. R. N. Bracewell, *The Fourier Transform and Its Applications*, 3rd ed. McGraw-Hill, 2000.

Tradeoff robustezza-accuratezza

45. D. Tsipras, S. Santurkar, L. Engstrom, A. Turner, and A. Madry, "Robustness may be at odds with accuracy," in *Proc. ICLR*, 2019.
46. E. Raghunathan, S. M. Xie, F. Yang, J. Duchi, and P. Liang, "Understanding and mitigating the tradeoff between robustness and accuracy," in *Proc. ICML*, 2020.

47. H. Zhang *et al.*, “Theoretically principled trade-off between robustness and accuracy (TRADES),” in *Proc. ICML*, 2019.

Testi di riferimento

48. I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016.
49. S.-I. Amari, *Information Geometry and Its Applications*. Springer, 2016.
50. C. E. Shannon and W. Weaver, *The Mathematical Theory of Communication*. University of Illinois Press, 1949.
51. D. P. Bertsekas, *Dynamic Programming and Optimal Control*, 4th ed. Athena Scientific, 2017.
52. R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*, 2nd ed. MIT Press, 2018.
53. C. Dwork and A. Roth, “The algorithmic foundations of differential privacy,” *Foundations and Trends in Theoretical Computer Science*, vol. 9, nos. 3–4, pp. 211–407, 2014.

Mathematics of Neural Networks — Volume IV of V
Silvano Salvador